Dreidimensional speicherintegrierte Verarbeitungsarchitekturen: Ein holistisches Werkzeug zur Modellierung und Simulation

Von der
Carl-Friedrich-Gauß-Fakultät
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades eines
Doktoringenieurs (Dr.-Ing.)

genehmigte
Dissertation

von
Dipl.-Inf. Patrick Daniel Marcus Siegl
geboren am 14. Mai 1986
in Stuttgart

Eingereicht am:	22. März 2018
Disputation am:	28. August 2018
1. Referent:	Prof. Dr.-Ing. Mladen Berekovic Universität zu Lübeck
2. Referent:	Prof. Dr.-Ing. Harald Michalik Technische Universität Carolo-Wilhelmina zu Braunschweig

2018

Eidesstattliche Erklärung

Dipl.-Inf. Patrick Daniel Marcus Siegl
Leisewitzstr. 30
30175 Hannover

Ich erkläre hiermit an Eides statt,

- dass ich die vorliegende Dissertation mit dem Thema

  Dreidimensional speicherintegrierte Verarbeitungsarchitekturen: Ein holistisches Werkzeug zur Modellierung und Simulation

  selbständig verfasst, keine Textabschnitte von Dritten oder aus eigenen Prüfungsarbeiten ohne Kennzeichnung übernommen und alle von mir benutzten Hilfsmittel und Quellen angegeben habe,
- dass Dritte weder unmittelbar noch mittelbar geldwerte Leistungen von mir für Vermittlungstätigkeiten oder für Arbeiten erhalten haben, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen,
- dass ich die vorliegende Dissertation noch nicht veröffentlicht habe,
- dass ich die Dissertation noch nicht als Prüfungsarbeit für eine staatliche oder andere wissenschaftliche Prüfung eingereicht habe,
- dass ich noch kein Promotionsgesuch gestellt habe,
- dass ich die gleiche, eine in wesentlichen Teilen ähnliche oder eine andere Abhandlung nicht bei einer anderen Hochschule als Dissertation eingereicht habe,
- dass mir die geltende Promotionsordnung der Fakultät bekannt ist
- und dass ich die Grundsätze zur Sicherung guter wissenschaftlicher Praxis an der Technischen Universität Braunschweig kenne und beachtet habe.

Hannover, den 8. September 2018

Dipl.-Inf. Patrick Daniel Marcus Siegl

Vorwort

Eine Promotion ähnelt einer langen Reise in die Ferne. Ferne Ziele gespickt mit unbekannten Wahrnehmungen erwirken neue Erkenntnisse und inspirierende Einfälle. Auf solch einer Reise kommt die Dissertation einem Tagebuch gleich, welches gewonnene Eindrücke und gemachte Erfahrungen nicht nur widerspiegelt und strukturiert, sondern anhand einer innovativen Idee zugleich vervollständigt. Auf meiner Reise konnte ich viele Begegnungen wahrnehmen, welche mich unterstützt haben und ohne die sich die erbrachte Dissertation in diesem wissenschaftlichen Kontext nicht ergeben hätte. Vielen dieser Begegnungen bin ich zu großem Dank verpflichtet, sodass dies im Folgenden anerkannt werden soll.

Mein Dank geht an Prof. Dr.-Ing. Mladen Berekovic, welcher mich als Promotionsstudenten mit umfangreichem Freiraum betreut und mit großem Vertrauen an das IBM T.J. Watson Research Center empfohlen hat. Gleicher Dank gebührt Dr. rer. nat. Rainer Buchty, welcher durch sein Handeln für mich Weichen stellte, ohne die weder diese Dissertation noch die gemeinsamen wissenschaftlichen Veröffentlichungen zustande gekommen wären. Den wissenschaftlichen Austausch in die USA initiiert hat und somit nicht unerwähnt bleiben soll Jamin Naghmouchi.

Für Anregungen, Impulse und den unermüdlichen Anstoß zu dieser Dissertation möchte ich der Abteilung Exascale System Software bzw. Data Centric System Software des IBM T.J. Watson Research Center danken, welche mir einen allumfassenden, nur äußerst schwer anderweitig möglichen Einblick in die Thematik der Höchstleistungsrechnerarchitekturen bot. Insbesondere möchte ich meinen dortigen Mentor M.Sc. Jose Brunheroto hervorheben, welcher mich als seinen 10. bzw. 13. einzig nicht brasilianischen Promotionsstudenten großartig im genannten Bereich betreute. Jose ermöglichte mir ein äußerst tiefes Eintauchen in den Entwurf und die Modellierung von Prozessor-/Speicherarchitektursimulatoren – im spezifischen bei den von uns entwickelten Simulationsmodellen Active Memory Cube (AMC) sowie den beiden Aggregated Global Extended Memory (AGEM) Global Address Space (AGEM GAS) und Coordination Namespace (AGEM CNS) –, welches alleinig durch eine Danksagung nur schwer zu honorieren ist. Inspiriert durch die Gespräche mit Dr. Ravi Nair über PIM, trug auch er mit seinen konzeptionellen Vorschlägen zu dieser Dissertation bei.

Stets an meiner Seite, jedoch für sie oft nicht einfach, begleitete mich meine Familie auf meiner Reise. Für die finanzielle Unterstützung derer gilt meiner Mutter Irene Dank. Gleichfalls bin ich meiner langjährigen, großen Liebe und Frau Janine dankbar, welche mir trotz Distanz mehrere Jahre den USA-Aufenthalt ermöglichte und somit einen signifikanten Freiraum zur wissenschaftlichen Entfaltung schuf.

Im Andenken an alle die, die mir lieb und wichtig waren.

Patrick Siegl

Kurzfassung

Die gemeinhin als *Memory Wall* bekannte, sich stetig weitende Leistungslücke zwischen Prozessor- und Speicherarchitekturen erfordert neue Konzepte, um weiterhin eine Skalierung der Rechenleistung zu ermöglichen. Da Speicherarchitekturen als die beschränkende Komponente innerhalb einer *Von-Neumann-Architektur* identifiziert wurden, widmet sich die vorliegende Arbeit dieser Problemstellung. Dabei werden flüchtige und nichtflüchtige Speicherarchitekturen illustriert sowie Techniken zur Vermeidung von Latenzen und zur Verbesserung der Bandbreite erarbeitet. Obgleich neuartige Speicherarchitekturen, welche etwa die dreidimensionale Integration mit *Through-Silicon Vias* (TSV) nutzen, zu einer Linderung der *Memory Wall* beitragen werden, sind diese für die zukünftige Skalierung ungenügend. Ein vielversprechender Ausweg stellt die Integration von Rechenkapazität in den Speicher dar, wodurch die Effizienz aufgrund von naher, ungebundener Bandbreite und geringer Latenz signifikant verbessert wird, weshalb dies bereits in der Vergangenheit im Rahmen des planaren, speicherintegrierten Verarbeitens (engl.: *Processing-In-Memory*, PIM) untersucht wurde. Entsprechend illustriert diese Arbeit PIM sowie das *Near-Memory Processing* (NMP) als Varianten von Architekturen, welche unter der Begrifflichkeit der datennahen Rechenverarbeitung (engl.: *Near-Data Processing*, NDP) zusammengefasst sind. Wohingegen historische PIM noch planar mit einer einheitlichen Prozesstechnologie integriert wurden und dadurch technische Nachteile aufwiesen, können diese durch das dreidimensionale Stapeln abgebaut werden, weshalb neuartige PIM exzellente Kandidaten für die zukünftige Skalierung von Rechenleistung darstellen.

Trotz großem Potential weist die vorliegende Arbeit einen Mangel an geeigneten PIM-Simulationsmodellen für eine Entwurfsraumexploration auf, was aus der Komplexität also dem internen, als heterogenes Rechensystem identifizierbaren, Aufbau resultiert. Daher wurde ein parametrisierbares Simulationswerkzeug für dreidimensionale Speicherstapel geschaffen, welches durch die Fähigkeit zur Modellierung von speicherintegrierten Verarbeitungsarchitekturen erweitert wurde. Aufgrund der hohen Flexibilität und Unterstützung von externem Routing kann dieser Speicherstapel etwa den *Hybrid Memory Cube* nach der Spezifikation 2.1 standardkonform, auch innerhalb einer Multi-HMC-Umgebung, simulieren. Zugleich bietet es eine hohe Simulationsgenauigkeit indem auf elementaren Datenpaketen in Kombination mit dem Hardware validierten Simulator *BOBSim* modelliert wird, wohingegen eine schnelle Simulationsausführung bei Bedarf mittels eines eigens entworfenen, effizienten Simulationstaktbaums hinzuschaltbar ist. Messungen weisen hierzu im funktionalen Modus eine Beschleunigung um den Faktor 100 gegenüber traditionellen Ansätzen auf, wohingegen eine Verdoppelung der Ausführungsgeschwindigkeit mit Taktgenauigkeit erzielt wird.

Zusätzlich zum entworfenen Simulationswerkzeug wird innerhalb der Arbeit die Integrationsfähigkeit dessen hin zur Modellierung einer vollständigen speicherintegrierten Verarbeitungsarchitektur mithilfe eines Beschleunigers demonstriert. Hierfür wird ein Simulationsmodell einer Speicherlatenz unempfindlichen, zur *NYUZI*-ISA binärkompatiblen Grafikkartenarchitektur ent-

wickelt, Hardware validiert und vertikal in das Simulationswerkzeug integriert. Die maximal verfügbaren Hardwareressourcen für die verschiedenen Konfigurationen aus GPU-Rechenkernen und HMC-Speicher orientieren sich wiederum an einem ähnlichen PIM-Beschleuniger aus der Literatur. Evaluiert wird einerseits das GPU-Simulationsmodell eigenständig, andererseits als PIM-Verbund jeweils mit Hilfe einer repräsentativ gewählten, speicherbeschränkten geophysikalischen Bildverarbeitung. Diese beruht auf einer in der Öl- und Gasindustrie üblichen seismischen Reverse Time Migration (RTM), welche als zweidimensional, vorwärtsgerichtete RTM mit einem 9-Punkt-*Stencil* zweiter Ableitung und vierter Ordnung betrieben wird.

Bei alleiniger Betrachtung des GPU-Simulationsmodells weist dieses eine signifikant gesteigerte Simulationsgeschwindigkeit auf, welche sich im Bereich eines Emulators wiederfindet. Gleichzeitig und trotz statischer Speicherlatenz stellt sich gerade einmal eine Abweichung von 6 % gegenüber dem Verilator-Modell[1] ein. Nachfolgend werden innerhalb dieser Arbeit unterschiedliche Konfigurationen des integrierten PIM-Beschleunigers evaluiert. Je nach gewählter Konfiguration kann danach die seismische Reverse Time Migration entweder bis zu 140 GFLOPS an tatsächlicher Rechenleistung abrufen oder eine maximale Recheneffizienz von synthetisch 30 % bzw. real 24,5 % erzielen. Letzteres stellt eine Verdopplung des Stands der Technik dar, obgleich rein die Speicherhierarchie auf speicherintegrierte Verarbeitung nicht jedoch die Instruktionssatzarchitektur (ISA) selbst tief angepasst wurde. Eine anknüpfende Diskussion erläutert eingehend die Resultate.

[1] Der Verilator ist ein frei verfügbares Werkzeug zur Konvertierung von (System)Verilog in taktgenaue C++- oder SystemC-Modelle.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Eine Million Fließkommaoperationen im Jahr 1964 (CDC 6600; MegaFLOPS), eine Milliarde im Jahr 1985 (Cray-2; GigaFLOPS), eine Billion im Jahr 1996 (Intel ASCI Red; TeraFLOPS) und eine Billiarde im Jahr 2008 (IBM Roadrunner; PetaFLOPS): Die Halbleiter- und die darauf aufbauende Computertechnologie wiesen innerhalb der letzten 50 Jahre eine Skalierung um einen Größenfaktor von einer Milliarde auf. Demnach müsste in den Jahren 2018 bis 2020 ein Höchstleistungsrechensystem die Marke ExaFLOPS d.h. eine Trillion Fließkommaoperationen pro Sekunde überschreiten. Obgleich ExaFLOPS eine rein technische Hürde ist, stellt die anhaltende Skalierung eine außerordentliche Herausforderung dar. Hintergrund sind vielfach ausgeschöpfte Mittel zur Entwicklung von neuartigen Prozessorarchitekturen sowie die vielen als *Brick Wall* bekannt gewordenen Einschränkungen, darunter die *Memory Wall*, *Bandwidth Wall*, *ILP Wall*, *Power Wall* und neuerdings die *Locality Wall*. Zeitgleich nur wenig Beachtung geschenkt, jedoch als integraler Bestandteil einer ganzheitlichen *Von-Neumann-Architektur* die Rechenleistung signifikant beeinflussend, bieten Speicherarchitekturen aufgrund deren Optimierung auf Kapazität und Kosten, hohen Stromhunger, geringe Speicherbandbreite und hohe Latenz, sodass ein akuter Bedarf an Forschung besteht. Ebenfalls stößt die darauf aufbauende Speicherhierarchie im Zeitalter von vielen dezentralisierten Recheneinheiten bedingt durch Komplexität und schwierige Realisierbarkeit an ihre Grenzen. Neue visionäre Ideen wie der daten- statt prozessorzentrische Entwurf von Höchstleistungsrechensystemen, sowie die Effizienzsteigerung durch mehr Lokalität mit weniger ineffizienten Datentransfers adressieren hierbei ausdrücklich Speicherarchitekturen, wobei diese mit gängigen Verarbeitungsarchitekturen nur ungenügend umgesetzt werden können. Revolutionär wäre eine Auslagerung von Rechenkapazität in den Speicher (Re-Integration), wodurch sich die Lokalität im Idealfall signifikant erhöhen ließe, um darüber eine starke Skalierung der Rechenleistung zu ermöglichen. Neue Ansätze wie etwa das dreidimensionale Stapeln sowie Programmierparadigmen für heterogene Rechensysteme könnten dabei die in den 1960-er und 1970-er Jahren aufgezeigten technischen Problemstellungen überwinden, sodass primär Aufgabenstellungen auf datennahe Recheneinheiten und nicht mehr Daten zu verschieben wären. Jedoch mangelt es an geeigneten Simulationswerkzeugen um speicherintegrierte Verarbeitungsarchitekturen erfolgreich ganzheitlich zu modellieren und um innovative Einsichten zu ermöglichen. Dies resultiert aus der Architektur, welche ein komplex heterogenes System aus Speicher- und Verarbeitungsarchitektur darstellt, weshalb ein jeweiliges Simulationsmodell zur Integration benötigt wird. Entsprechend widmet sich diese Arbeit dem Entwurf eines Werkzeugs zur Modellierung und Simulation eines dreidimensionalen Speicherstapels, welcher um die Fähigkeit von

speicherintegrierter Rechenbeschleunigung erweitert wurde.

Höchstleistungsrechensysteme und die damit einhergehenden, genannten großen Zahlen mögen befremdlich sein, da sie kaum begreiflich noch realisierbar gemacht werden können. Jedoch wird diese Entwicklung – wie auch zuvor – unser aller Leben beeinflussen und verändern. Einerseits, da die alte Generation an Höchstleistungsrechnern in der heutigen Form u.a. als Smartphone, Tablet und Laptop unseren Alltag als unsere treuen Begleiter bestreiten. Andererseits, da das stetige Wachstum Problemstellungen in den Fokus bringt, welche bisher aufgrund hoher Anforderungen an die Rechenleistung zu komplex oder schlicht nicht berechenbar waren. Ausgemachte Anwendungsgefilde für den aktuellen ExaFLOPS-Bedarf sind insbesondere im technisch-wissenschaftlichen sowie kommerziellen Bereich wie etwa der Nuklear- und Windenergie, Klima, (Motoren-) Verbrennung, CO2-Sequestrierung, chemische Wissenschaft, Fertigungsmethoden, Medizin, Kosmologie und Astrophysik, Geologie sowie Seismik und Strömungsmechanik vorzufinden [Messina, 2017]. Viele der genannten Anwendungsgebiete besitzen Aspekte, die nicht effizient oder nur mit exponentiellen Rechenaufwand berechnet werden können, weshalb in der theoretischen Informatik diese in die Klasse der nichtdeterministisch in Polynomialzeit lösbaren Probleme (NP-vollständig) fallen. Obgleich weit gestreut, beeinflussen alle genannten Themen auch in der ein oder anderen Form indirekt oder direkt die Art des Lebens und somit den privaten Bereich. Entsprechend stellt die weitere Skalierung der Rechenleistung insbesondere die aktuelle ExaFLOPS-Hürde eine Angelegenheit nationaler bis internationaler Tragweite dar. Unabhängig voneinander finanzieren daher die Vereinigten Staaten von Amerika [Messina, 2017], Europa [ETP4HPC, 2017, European Exascale Projects (FP7), 2016], China [InsideHPC, 2017] und Japan [Ishikawa, 2015] Forschungsprojekte unter der Begrifflichkeit Exascale, nicht nur zur Entwicklung von entsprechenden Höchstleistungsrechensystemen sondern auch zur Identifikation von Exascale-Problemstellungen, welche auf derartige Höchstleistungsrechnersysteme angewiesen sind.

1.1 Motivation

Neu in den Fokus rückende Problemstellungen sind häufig beschränkt, soll heißen limitiert durch ihre speicherintensive Charakteristik, wodurch eine Skalierung der Rechenleistung primär von der eingesetzten Speichertechnologie abhängig ist. Zumeist resultiert solch speicherintensive Charakteristik aus der Nutzung von dünn besetzten Datensätzen, welche eine äußerst schwierig vorhersagbare Regularität aufweisen, oder durch Strömungsalgorithmen, welche alte Daten kaum bis gar nicht wiederverwenden. Entsprechend nimmt die Bedeutung eines einzelnen Rechenkerns hinsichtlich dessen architektonischen Aufbau sowie dessen Verarbeitungsgeschwindigkeit ab, obgleich dieser die tatsächliche Rechenkraft zur Verfügung stellt. Wohingegen durchsatz- sowie latenzoptimierte Architekturen (Grafikkarten, DSPs, $\cdots$) die Auswirkungen der *Memory Wall* als auch von derartigen Anwendungen zwar nicht vermeiden, jedoch lindern können, so stellt dies für gängige CPU-Architekturen wiederum ein signifikantes sowie stetig größer werdendes Problem dar. Exemplarisch ist ein Auszug von solch aktuell besonders groß herausgebrachten Anwendungsgefilde in Tabelle 1.1 aufgeführt. Dieser umfasst den mit viel Forschungsaufwand versehenen Bereich der neuronalen Netzwerke, inklusive künstliche Intelligenz und maschinelles Lernen. Danach sind die meisten der verwendeten Anwendungen beschränkt durch die Speichergeschwindigkeit.

Klassifikation	Beispiel
Rein rechengebunden	Mathematische Optimierung Monte-Carlo-Simulation
Rechengebunden, Netzwerk- / speichergebunden	Zeitreihenanalyse
Vornehmlich speichergebunden, I/O-gebunden (Platte)	Text Mining Regressionsanalysen Clusteranalyse Nearest-Neighbor-Heuristik Neuronale Netzwerke Support Vector Machine Empfehlungsdienste
Speichergebunden, I/O-gebunden (Platte)	Online Analytical Processing Graphentheorie Hermeneutik Entscheidungsbäume Assoziationsanalyse

Tab. 1.1: Leistungsklassifikation von Methoden, welche bei analytischen Informationssystemen (engl.: *analytics*) Anwendung finden [Bordawekar et al., 2014, Bordawekar, 2014].

1.1.1 Speicherarchitekturen und speicherintegrierte Rechenverarbeitung

Innovative Anwendungen mit starker speicherintensiver Charakteristik, *Memory Wall* und *Bandwidth Wall* machen es notwendig, aktuelle Speicherarchitekturen sowie die Ausnutzung von Lokalität zu überdenken. An sich erscheint die Problematik der Speicherarchitekturen befremdlich, wiesen doch in der Vergangenheit Wissenschaftler wie Elliott et al. sowie Kogge auf die innerhalb eines DRAMs parallel geschalteten Stromverstärker und den Zeilenpuffer hin, an welchen jeweils eine signifikante Speicherbandbreite verfügbar ist [Elliott et al., 1992, Kogge, 1994]. Gleichzeitig jedoch erkannten beide an, dass bis zum Erreichen des Rechenkerns – zugespitzt – bis zu 99 % davon verschwendet wird [Kogge, 1994]. Entsprechend existieren Hierarchien aus unterschiedlichen Speichertypen zur Milderung, welche jedoch nur einen Bruchteil dessen anhand von Lokalität wiederherstellen [Kogge et al., 1995]. Trotzdem und gerade deshalb gelten aktuelle Speicherarchitekturen als äußerst langsam, energiehungrig und innerhalb von Höchstleistungsrechnern als teuerste und fehleranfälligste Komponenten. Im Zeitalter der Effizienzsteigerung, insbesondere im Höchstleistungsrechnen getrieben durch harte Stromkonsumvorgaben, wie etwa das durch das Energieministerium der USA (engl.: *US Department of Energy*, DoE) gesetzte Budget von maximal 20 Megawatt [Bergman et al., 2008], und in Anbetracht des Überwindens der ExaFLOPS-Hürde stellt dies einen unbefriedigenden Zustand dar. Zusätzlich besteht eine langsame Adaptierung von neuen Speichertechnologien und Speicherfunktionalität, wodurch noch immer umständlich RAS-, CAS- und PreCharge-Befehle Anwendung finden, obgleich moderne Protokolle abstrakte, einfach zu implementierende Befehle wie READ oder WRITE ermöglichen könnten. Letzteres könnte zu mehr Funktionalität und entsprechend Intelligenz in Speichern führen und hätte das Potential, NUMA-Systeme zu verbessern, bei welchen trotz unterschiedlicher Speicherdomänen, Speichertechnologien, Speicherprotokolle und Speicheranschlüsse Kohärenz gefordert wird.

Ein desaströses Bild, jedoch erscheint mit der industriellen Fertigungsmöglichkeit von *Through-Silicon Vias* (TSV) ein plausibler Ausweg gelungen zu sein, indem diese eine enge dreidimensionale Vernetzung von unterschiedlichen Prozesstechnologien und Materialien erlauben [Motoyoshi, 2009]. Somit kann eine Re-Integration von Rechenkapazität in Speicherarchitekturen erfolgen, wodurch eine höhere Lokalität erzielt werden kann. Ähnliche planare Ansätze in den 1970-er und 1990-er Jahren, zumeist bedingt durch ein fehlendes Geschäftsmodell nur in der Literatur als Studien erarbeitet, illustrierten reduzierte Speicherlatenz und niedrigen Energiekonsum in Kombination mit hoher Speicherbandbreite. Allerdings litten die meisten durch die zu wählende Prozesstechnologie, weshalb entweder eine kleine Speicherkapazität bei signifikanter Rechenleistung oder eine geringe Rechenleistung mit großer Speicherkapazität ermöglicht wurde. Durch den Einsatz von TSVs überwindet etwa Microns kommerzielle Ausführung namens *Hybrid Memory Cube* (HMC) vergangene Problemstellung indem mehrere Speicherebenen mit einer Logikebene vertikal integriert werden, wobei gleichzeitig ein modernes, abstraktes Speicherprotokoll mit einer einheitlichen Speicherschnittstelle gegeben ist [Pawlowski, 2011]. Entsprechend begann nach der Vorstellung HMCs eine Hochphase der Forschung im Bereich der speicherintegrierten Verarbeitung (engl.: *processing-in-memory*, PIM), um die erwartete erhöhte Speicherbandbreite sowie reduzierte Speicherlatenz auszuschöpfen.

Obgleich viel Forschung im Bereich von planaren und dreidimensionalen PIM-Architekturen betrieben wurde und wird, ist im spezifischen für letzteres noch unklar, welche Architekturen für Exascale-Problemstellungen am profitabelsten sind. Gerne wird aufgrund der evolutionären, dreidimensionalen Integration die *Von-Neumann-Architektur* in Frage gestellt, würde dies mit der gleichzeitigen Vermeidung des *Von-Neumann-Engpass* einer Revolution gleichen [Dlugosch et al., 2014]. Eine Evaluierung von solch modernen Ideen gestaltet sich als äußerst schwieriges Unterfangen, da innerhalb dieser Nische kaum fähige Modellierungswerkzeuge für hochparallele PIM-Architekturen zur Verfügung stehen. Dies resultiert unweigerlich nicht nur aus den kaum verfügbaren und meist ungenügend ausgestatteten, dreidimensionalen Speicherstapelmodellen, sondern auch aus den kaum gegebenen Beschleunigermodellen. Gerade deshalb besteht eine Lücke für eine ganzheitliche Entwurfsraumexploration von PIM-Architekturen, welche diese Arbeit zu schließen vermag.

1.1.2 Exascale-Problemstellung: Seismisch geophysikalische Bildverarbeitung

Abgesehen von modernen Anwendungsdomänen trifft die *Memory Wall* auch die traditionellen Problemstellungen innerhalb des Höchstleistungsrechnens (engl.: *high performance computing*, HPC). Der aufgrund seiner tatsächlich nutzbaren Baukastenmethoden beliebteste Vertreter dieser Nische, namhaft der *High Performance Linpack*-Benchmark, ist aufgrund seiner Rechengebundenheit weniger betroffen. Andere wichtige Anwendungen des Höchstleistungsrechnens, – wie etwa die u.a. durch die Exascale Initiative hervorgehobene, als auch die als erste Exascale-Problemstellung definierte Seismik (Abbildung 1.1) [Morton et al., 2011, Gara, 2012], – weisen durch die Nutzung von geophysikalischer Bildverarbeitung und somit häufigem Einsatz von *Stencil*-Implementierungen, eine äußerst hohe Speichergebundenheit entsprechend speicherintensive Charakteristik auf. Gerade *Stencil*-Implementierungen sind jedoch von außerordentlicher Wichtigkeit, nicht nur für die Öl- und Gasindustrie, sondern auch für viele hierauf aufbauende numerische Verfahren [Datta, 2009].

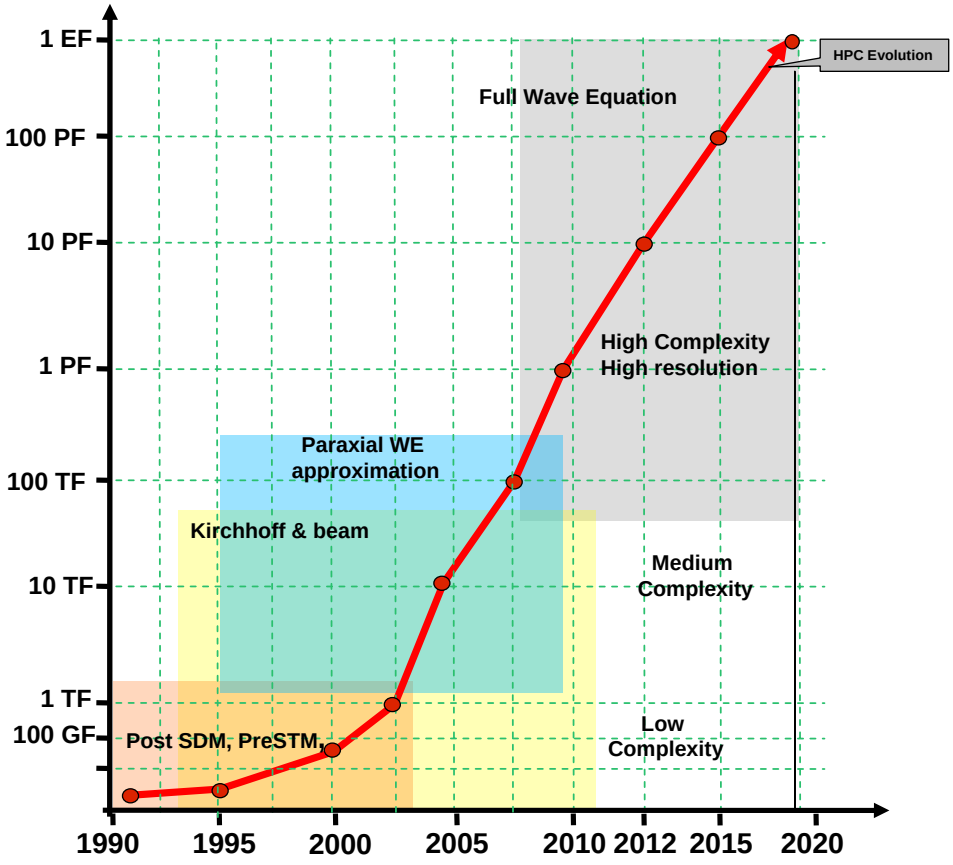


Abb. 1.1: Anstieg der Rechenkomplexität bedingt durch die Verwendung von verbesserter und höher auflösender, seismisch geophysikalischer Bildverarbeitung [Morton et al., 2011].

Beispielhaft soll das Experiment von Tsuboi et al. genannt werden, welches eine globale – im Sinne von die vollständige Erde (*SPECFEM3D GLOBE*) umfassend – seismische Wellensimulation auf dem mit 10,6 PetaFLOPS an Rechenleistung ausgestatteten Höchstleistungsrechner K Computer in Japan modelliert [Tsuboi et al., 2016]. Dabei wurden 11,84 % der Fließkommarechenleistung erzielt, obgleich der Quelltext eine parallele Effizienz von 99,54 % aufwies, dadurch hoch skalierbar war. Das Beispiel erscheint alarmierend angesichts dessen, dass Kahle und Odegard unabhängig voneinander einen für die Öl- und Gasindustrie typischen, bereits erhobenen und noch zu verarbeitenden, seismischen Datenbestand von multiplen Petabytes aufzeigen, welcher sich mit jeder weiteren Tiefseebodenuntersuchung jeweils um bis zu 660 Terabyte vergrößert [Kahle, 2017, Hemsoth, 2017a]. Ein ähnliches Szenario zeichnet Baker Hughes Vizepräsident für digitale Entwicklung Mathew, welcher hinzufügt das die Öl- und Gasindustrie für die kommenden Jahre rein zur Verarbeitung von Sensordaten zwingend auf ExaFLOPS angewiesen ist, wobei der laut ihm "explodierende" Bereich der geophysikalischen Bildverarbeitung hierbei nicht einmal mit eingerechnet ist [Hemsoth, 2018]. Entsprechend werden laut Kahle 80 % der gewonnenen Daten nicht genutzt, da zur geophysikalischen Bildverarbeitung von einer vollständigen Tiefseeprobe ein hundertfaches an Rechenleistung nötig wäre, obgleich bereits ein Datenauszug auf tausenden bis zehntausenden von Rechenkernen Wochen bis Monate an Rechenzeit benötigt [Kahle, 2017, Newman, 2010]. Zugleich beansprucht die Visualisierung der in Betracht gezogenen Daten Rechensysteme signifikant, da ultra-hochauflösende, dreidimensionale Bilder mit Hilfe von mehreren Terabyte umfassenden Datensätzen erstellt werden. Im direkten Vergleich stellt jedoch die geophysikalische Bildverarbeitung die Hauptlast dar [Grosser et al., 2011].

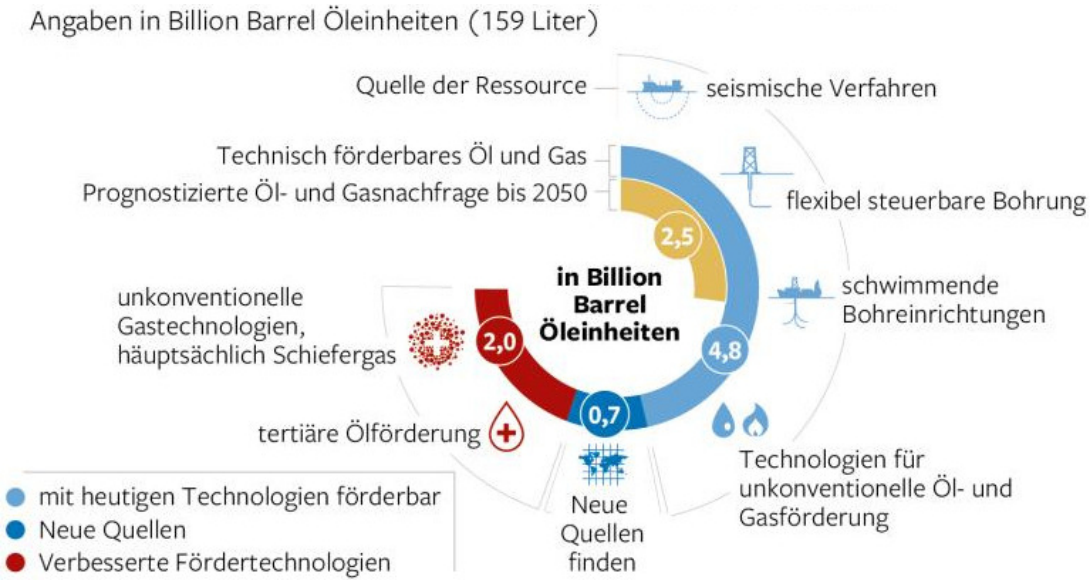


Abb. 1.2: BPs technologischer Erdölgewinnungsausblick für das Jahr 2050 [BP, 2015, Panitz und Trentmann, 2015].

Unabhängig von der großen Rechenherausforderung ist die geophysikalische Bildverarbeitung innerhalb der Seismik für die Öl- und Gasindustrie essentiell notwendig, um bei neuen Territorien eine präzise Vorhersage generieren zu können. Entsprechend errichtete der Energiekonzern Eni im Jahr 2018 hierfür einen neuen, privat betriebenen Höchstleistungsrechner mit 18,6 PetaFLOPS[1] [ENI, 2018]. Unabhängig davon gab der Erdölgigant BP in seinem technologischen Ausblick

[1] Theoretisch Platz 5 in der Top500 Liste von November 2017 [Meuer et al., 2014].

an, dass in den 1990-er Jahren der Einsatz von seismisch geophysikalischer Bildverarbeitung die Vorhersagegenauigkeit von ehemals 30 % auf 50 % steigerte, wobei modernere Verfahren wie etwa die Hinzuziehung von Zeit innerhalb eines vierdimensionalen Modells dies weiter steigern lässt [BP, 2015]. BP selbst geht von einer anhaltenden Nachfrage nach Öl und Gas aus, welche demnach um 35 % bis in das Jahr 2050 zunehmen wird [BP, 2015]. Nicht alleinig, aber als Schlüsseltechnologie innerhalb eines Spektrums an weiteren Verfahren, soll dies mit Hilfe von seismisch geophysikalischer Bildverarbeitung laut BP bedienbar sein (Abbildung 1.2).

1.2 Ziele und Forschungsbeiträge

Für die vorliegende Arbeit wurden vier Ziele definiert, welche sich der Schaffung eines flexiblen und leistungsfähigen Simulationswerkzeugs zur Modellierung von dreidimensionalen Speicherstapeln wie *Hybrid Memory Cube* sowie den *Processing-In-Memory*-Rechenbeschleunigern widmen:

- Entwicklung eines speicherbandbreiten-akkuraten Simulationsmodells für einen generisch dreidimensionalen Speicherstapel, vornehmlich zur Modellierung eines *Hybrid Memory Cube* (HMC) gemäß Spezifikation 2.1.
- Spezifikation und Modellierung eines Rechenbeschleunigerkerns für eine nachfolgende Integration in eine *Hybrid Memory Cube*-Architektur.
- Integration beider in eine flexible und leistungsfähige PIM-Simulationsumgebung zur Erforschung von PIM-Rechenbeschleunigungseffekten am Beispiel von HMC.
- Validierung der Arbeit anhand eines aktuellen, für das Höchstleistungsrechnen (engl.: *high performance computing*, HPC) wichtigen, repräsentativ gewählten Beispiels aus dem Bereich der seismisch geophysikalischen Bildverarbeitung.

Aus den gesteckten Zielen leiten sich drei konkrete Forschungsbeiträge ab:

- Entwicklung einer flexiblen, hochperformanten Simulationsumgebung für PIM-Architekturen am Beispiel von HMC.
- Exemplarischer Einsatz der entwickelnden Simulationsumgebung zur Entwurfsraumexploration von einer PIM-Beschleunigerarchitektur.
- Analyse von speicherintegrierter Rechenbeschleunigung auf speicherbandbreitenbeschränkte Strömungsalgorithmen demonstriert an einem repräsentativen Algorithmus aus der seismisch geophysikalischen Bildverarbeitung.

Der Kern der Arbeit stellt die Entwicklung eines mit hoher Flexibilität ausgestatteten, entsprechend parametrisierbaren Simulationswerkzeugs zur Modellierung von dreidimensionalen Speichern wie HMC dar. Sofern HMC als Speichermodell gewählt, ist dieses nach HMC-Spezifikation 2.1 standardkonform und modelliert auf elementarer Datenpaketebene (engl.: *flow control unit*, FLIT). Dadurch und durch die Integration des hardwarevalidierten Simulators *BOBSim* (engl.: *buffer-on-board*, BOB), welcher zur Modellierung unabhängiger, interner Speicherstapel (engl.: *vault*) innerhalb von HMC fungiert, bietet es eine größere Modellqualität mit

taktgenauer Simulation. Für das Routen von FLITs innerhalb von HMC wurde ein Mechanismus entwickelt, welcher zugleich die Vernetzung und das externe Routing zwischen mehreren HMCs in einer frei wählbaren Multi-HMC-Umgebung ermöglicht. Nach Aktivierung des eigens entwickelten, effizienten Simulationstaktbaums, erlaubt das entworfene Modell eine schnellere Simulationsausführung.

Anhand einer eingehenden, tiefen Literaturstudie der geschichtlichen Entwicklung von speichernahen und speicherintegrierten Verarbeitung in planaren und dreidimensionalen Speicherstrukturen, illustriert diese Arbeit hochprofitable PIM-Architekturen. Zusätzlich sind Mechanismen und Techniken geschildert, welche signifikante Rechenleistung erzielen können. Da bevorzugt spezialisierte Recheneinheiten hohe Speicherbandbreite verarbeiten und teils große Speicherlatenzen vermeiden können, wird ein adäquater Rechenbeschleuniger in der Gestalt einer latenztoleranten Grafikkarten-Architektur (engl: *graphics processing unit*, GPU) gewählt. Teilziel ist eine hardwaregenaue Implementierung dessen als binärkompatibles GPU-Simulationsmodell, welche eine schnell ausführende und taktgenaue Simulation, sowie eine aussagekräftige Charakterisierung bietet. Zusätzlich findet eine Integration des GPU-Simulationsmodells in das dreidimensionale Speichersimulationsmodell statt, um eine Entwurfsraumexploration und die Demonstration des flexiblen Simulationswerkzeugs zu ermöglichen.

Entsprechend befasst sich der letzte Beitrag mit der Validierung und der Demonstration des entwickelten PIM-Simulationswerkzeugs. Hierzu wird ein für die Öl- und Gasindustrie wichtiger Bereich innerhalb der als Exascale-Problemstellung identifizierten Seismik in Betracht gezogen, welcher Strömungseigenschaften besitzt. Im spezifischen wird repräsentativ ein in der seismisch geophysikalischen Bildverarbeitung genutzter und in der Industrie üblicher Reverse Time Migration (RTM) Algorithmus mit einem 9-Punkt-*Stencil* zweiter Ableitung und vierter Ordnung auf speicherintegrierte Beschleunigung hin analysiert. Inspiriert von zwischenspeicherlosen Prozessorarchitekturen findet dafür eine Modifizierung der Cachehierarchie des mit GPU-Architektur ausgestatteten PIM-Rechenbeschleunigers in Richtung hoher Rechenleistung bei gleichzeitiger Ressourcenknappheit statt, woraufhin das resultierende Konstrukt eingehend auf speicherintegrierte Beschleunigung, entsprechend Rechenleistung sowie Recheneffizienz untersucht wird.

1.3 Organisation der Arbeit

Die vorliegende Arbeit umfasst insgesamt acht Kapitel. Neben der Einleitung widmen sich vier Kapitel den Grundlagen und dem Stand der Technik, wohingegen sich jeweils ein weiteres Kapitel einerseits der Modellierung sowie andererseits der Auswertung annimmt. Daraufhin folgt eine Zusammenfassung mit Ausblick.

Im Detail vermittelt das nachfolgende Grundlagenkapitel fundamentale Gesetzlichkeiten und Begrifflichkeiten von Rechnerarchitekturen. Ebenso wird bereits die vollständige Algebra eingeführt, welche in der Evaluation zur Herleitung des verwendeten Algorithmus benötigt wird. Speicherarchitekturen und speicherintegrierte Verarbeitungsarchitekturen sind nicht Teil des allgemeinen Grundlagenkapitels, sondern erhielten als Schwerpunkt dieser Arbeit zwei eigenständige Kapitel. Das erste davon, namentlich das Kapitel der Speicherarchitekturen befasst sich ausgiebig mit gängigen planaren sowie dreidimensionalen Speichern, welche entweder flüchtig oder nichtflüchtig ausgeführt sind. Das zweite Kapitel wiederum widmet sich den speicherintegrierten sowie -nahen Verarbeitungsarchitekturen, welche physikalisch auf den Speicherarchitekturen

aufbauen. Dabei werden Begrifflichkeiten wie etwa NDP, PIM und NMP eingeführt und anhand einer Taxonomie eingeordnet. Zugleich wird beginnend mit den 1960-er Jahren ein tiefer Einblick in die geschichtliche Entwicklung von speicherintegrierten sowie -nahen Verarbeitungsarchitekturen im Rahmen von planaren und dreidimensionalen Speicherstrukturen gewährt.

Basierend auf der Schlussfolgerung, dass PIM ein komplexes Konstrukt aus Speicher- und Prozessorarchitektur darstellt, schlüsselt Kapitel fünf den Stand der Technik an Modellierungswerkzeugen für beide Aspekte auf. Entsprechend werden Werkzeuge zur Simulation und Emulation von GPGPU-Rechenbeschleunigern, Speicherarchitekturen und heterogenen Systemarchitekturen aufgezeigt, welche jeweils für eine speicherintegrierte Simulationsumgebung von Belang sind. Ausgehend von der Erkenntnis, dass weder ein adäquates Simulationsmodell für dreidimensionale Speicherstapel noch für Beschleunigerarchitekturen, geschweige denn ein vollständiges PIM-Simulationswerkzeug existiert, schildert Kapitel sechs die Modellierung dieser sowie die Integration zu einem vollständigen PIM-Simulationswerkzeug. Jedes der entwickelten Simulationsmodelle wird im Auswertungskapitel sieben präsentiert und auf die individuellen Alleinstellungsmerkmale hin evaluiert. Zusätzlich findet eine Implementierung des Algorithmus mit spezifischer Beschleunigung statt, welche zeitgleich zur Demonstration des intendierten PIM-Beschleunigers und zur Validierung genutzt wird. Eine Zusammenfassung der erbrachten Arbeit sowie einen Ausblick auf potentiell nachfolgende Tätigkeiten gibt Kapitel acht.

Kapitel 2

Grundlagen

Das im Folgenden erläuterte Kapitel der Grundlagen widmet sich einerseits dem Bereich der Rechnerarchitekturen, sowie andererseits den seismischen Berechnungsverfahren.

Ersteres beleuchtet elementare Aspekte sowie Problemstellungen, welche innerhalb von Rechnerarchitekturen zur Anwendung kommen oder denen sich die genannten fügen müssen. Obgleich Rechnerarchitekturen aus einer zentralen Verarbeitungs- und einer Speicherarchitektur bestehen, wird innerhalb diesen Kapitels primär der Fokus auf die zentrale Verarbeitungskomponente gelegt. Dies beruht darauf, dass historisch die Skalierung der Rechenleistung primär durch diese vorangetrieben wurde. Zusätzlich ist der Schwerpunkt dieser Arbeit die Re-Integration von Recheneinheiten in den Speicher, weshalb sich explizit zwei dedizierte, nachfolgende Grundlagenkapitel mit der Thematik Speicherarchitektur sowie speicherintegrierte Verarbeitungsarchitekturen beschäftigen.

Wie angedeutet ist ein weitere Teil der Grundlagen ist die Seismik. Da für die Evaluierung der zu erarbeitenden speicherintegrierten Verarbeitungsarchitektur ein repräsentativ gewählter, seismischer Algorithmus genutzt wird, soll innerhalb diesen Kapitels die elementare Algebra vermittelt werden. Zusätzlich wird ein grober Einblick auf verfügbare, seismische Werkzeuge sowie Algortihmensammlungen gegeben, welche wie im Falle von *Seismic Un*x* genutzt werden.

2.1 Rechnerarchitekturen

Der Entwurf von Rechner- sowie Speicherarchitekturen ist eine ingenieurwissenschaftliche Tätigkeit, welche dazu dient, unterschiedlichste Anwendungsszenarien unter gegebenen Rahmenbedingungen mit Hilfe einer Kompromissfindung maschinell zu beschleunigen. Trotz der heutigen Vielfalt an Anwendungsgebieten, in welchen solche Rechner- und Speicherarchitekturen vorzufinden sind, und auch der unterschiedlichen Größe von Architekturen, d.h. von kleinsten, eingebetteten Prozessoren, etwa für das *Internet-of-Things* (IoT), bis hin zu hochgradig parallelen Höchstleistungsrechnern (engl.: *supercomputer*), beruhen diese auf den gleichen Prinzipien, welche bereits im späten 19. jedoch primär im 20. Jahrhundert durch die Wegbereiter Babbage, Turing als auch Von-Neumann erdacht wurden. Bis zum heutigen Tag haben sich dabei grundlegende Gesetzlichkeiten, sowie Wahrnehmung für den Bereich der Rechnerarchitekturen herausgebildet, welche im Folgenden erläutert werden sollen.

2.1.1 Konsistenz und Kohärenz

Werden Prozessoren und Beschleuniger an einem gemeinsamen Bus mit einer gemeinsamen Speichersicht betrieben, so resultieren hieraus eine Konsistenz- sowie Kohärenz-Problematik, da die jeweiligen Recheneinheiten zumeist für kurze Speicherlatenzen mit dedizierten Zwischenspeichern ausgestattet sind. Solch Zwischenspeicher (engl.: *cache*) puffern entsprechende Lese- und Schreibzugriffe, wodurch letztere nicht zu den jeweiligen Nachbarspeichern propagieren. Ist ein Datum bereits in den Cache einer Nachbarrecheneinheit eingeladen, so vermag dieses auch bei einem *Write-Through*-Cache, welcher eine Speicheroperation bis zum Hauptspeicher durchreicht, nicht zu erkennen. Somit wird weiterhin ein altes Datum eingelesen.

Von Kohärenz wird gesprochen, wenn ein verteiltes Speichersystem bei jeglichem durch eine Recheneinheit initiierten Lesezugriff das aktuelle Datum zurückliefert, unabhängig davon, ob eine Nachbarrecheneinheit in seinen dedizierten Cache geschrieben hat. Trotzdem dürfen verteilte Caches für eine gewisse Adresse unterschiedliche Daten vorhalten, solange garantiert wird, dass bei einem Zugriff das zuletzt gespeicherte Datum zurückgeliefert wird. Um dies zu gewährleisten, muss nicht nur die Programmordnung eingehalten, sondern auch eine Schreibserialisierung in die jeweiligen Speicherzellen erzwungen werden.

Das Vorhalten von unterschiedlichen Daten in verteilten Caches zu einer gleichen Adresse ist demnach das Gegenteil von Konsistenz. Diese verlangt, dass alle verteilten Caches das gleiche Datum für die jeweilige Recheneinheit vorhalten, was technisch z.B. bedingt durch eine Prozessor-*Pipeline* bereits unmöglich ist. Somit stellt Kohärenz in Rechensystemen den Mechanismus dar, um eine Konsistenz zu suggerieren, obgleich diese tatsächlich nicht existiert. Ein gängiger Mechanismus zur Kohärenzwahrung ist das Bus-“schnüffelnde” (engl.: *snooping*) MESI-Cacheprotokoll, welches jeglicher Cachezeile innerhalb eines Caches einen von vier Zuständen zuweist: Modifiziert, Exklusiv, Geteilt (engl.: *shared*) und Ungültig (engl.: *invalid*). Diese werden je nach Situation verändert, um darüber eine kohärente Speichersicht zu gewährleisten.

2.1.2 Von-Neumann-Architektur

Im Jahr 1945 beschreibt Von-Neumann ein Grundkonzept eines Rechners, welches die Grundlage aller heutigen Computer darstellt. Diesen titulierte er als die *EDVAC*-Maschine [von Neumann, 1945]. Danach wird ein System in fünf Aspekte aufgeteilt, welche sich in eine zentrale arithmetisch-logische Einheit CA, ein zentrales Steuerwerk CC, verschiedene Formen an Speichern M, der Eingabe I, sowie der Ausgabe O gliedern (Abbildung 2.1). Zugleich weist er ein angeschlossenes Aufzeichnungsmedium R aus, welchem eine Diskussion folgt ob eine Kombination aus M und R benötigt wird oder gar nur einer der beiden ausreichend für die vorgestellte Architektur ist. Er schlussfolgert, dass beide Komponenten (“Organe”) benötigt werden, da R alleinig eine signifikante arithmetische Leistungseinbuße zur Folge hätte, sodass M trotz seiner Limitierungen zur Beschleunigung der Berechnungszeit führt. Bereits im *EDVAC*-Bericht – insbesondere in seinem Abschnitt 2.6 – motiviert Von-Neumann sein Konzept anhand des menschlichen Gehirns, was im Anbetracht von heutigen, architektonischen Denkansätzen wieder an Aktualität gewonnen hat (Unterabschnitt 3.3.2). Zugleich beschreibt Von Neumann die Verbindung zwischen den jeweiligen Organen bzw. Neuronen d.h. Komponenten als Synapsen, welche heutzutage als ein zentraler Bus bekannt ist.

Bedingt durch den zentralen Bus garantiert die *Von-Neumann-Architektur* eine sequen-

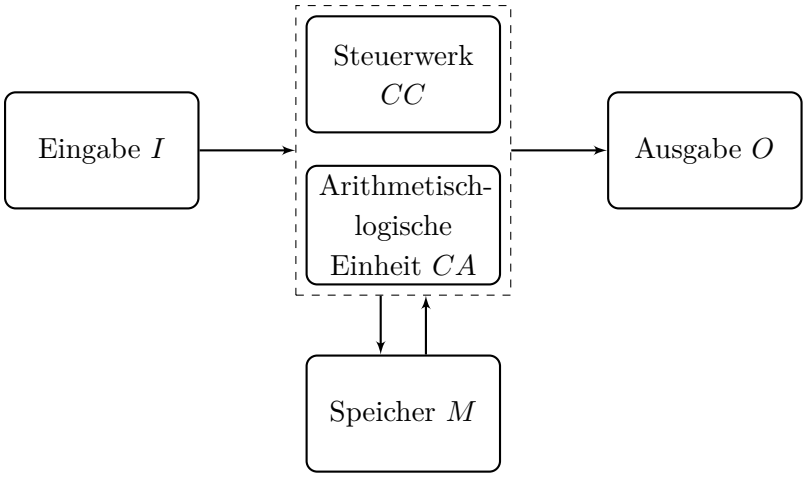


Abb. 2.1: Illustration einer *Von-Neumann-Architektur*.

tielle Befehlsabarbeitung, wodurch der von dem zentralen Steuerwerk ausgeführte Befehlsstrom aufgrund der vorhandenen Kohärenz weniger komplex ist. Obgleich die *Von-Neumann-Architektur* die Programmierung maßgeblich vereinfacht, führte sie einerseits den so genannten *Von-Neumann-Engpass* ein und ermöglichte andererseits die Entstehung der *Memory Wall* (Unterabschnitt 2.1.10). Der *Von-Neumann-Engpass* bezeichnet die einzelnen Verbindungen zwischen Speicher M und den beiden Komponenten Steuerwerk CC und der arithmetisch-logischen Einheit CA, welche gleichzeitig versuchen Befehle und Daten vom Speicher zu beziehen.

2.1.3 Harvard-Architektur

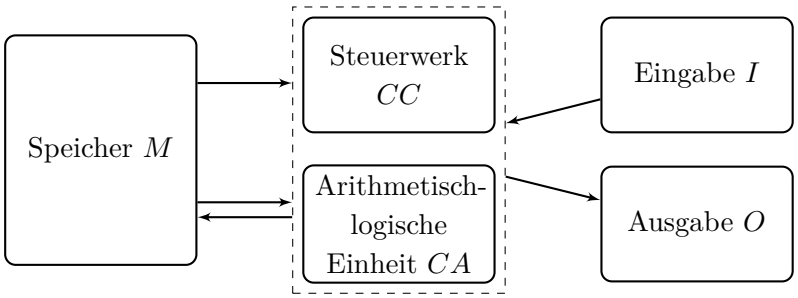


Abb. 2.2: Illustration einer *Harvard-Architektur*.

Zur Vermeidung des aus der *Von-Neumann-Architektur* resultierenden Engpasses sieht die *Harvard-Architektur* unabhängige parallele Verbindungen zum Speicher sowohl für das zentrale Steuerwerk als auch die arithmetisch-logische Einheit vor (Abbildung 2.2). Dies verhindert das Kreuzen von Befehlen, d.h. dem Befehlsstrom, und Daten, d.h. des Datenstroms, auf demselben zentralen Bus, jedoch führt dies gleichzeitig zu einem erhöhten Aufwand an nötigen elektrischen Verschaltungen sowie Komponenten. Heutige Rechnerarchitekturen nutzen meist eine modifizierte *Harvard-Architektur*. Hierbei wird innerhalb einer Hierarchie aus Zwischenspeichern (engl.: *caches*), der zur arithmetisch-logischen Einheit sowie zum zentralen Steuerwerk nächstliegende, meist als Level eins bezeichnete Zwischenspeicher (L1-Cache) als *Harvard-Architektur* separiert. Der Zwischenspeicher auf der Ebene zwei (L2-Cache), kann gleichsam in Befehls- und Datenzwischenspeicher (L2i- und L2d-Cache) aufgeteilt sein, jedoch ist dies nicht zwingend.

2.1.4 Klassifizierung nach Flynn

Befehle / Datenworte	SI*	MI*
*SD	SISD Single Instruction Stream Single Data	MISD Multiple Instruction Stream Single Data
*MD	SIMD Single Instruction Stream Multiple Data	MIMD Multiple Instruction Stream Multiple Data

Abb. 2.3: Die Klassifizierung nach Flynn.

Flynn definierte im Jahr 1966 eine zweidimensionale Klassifizierung, bei welcher die Anzahl der Befehls- und Datenströme unabhängig voneinander zu einem gegebenen Zeitpunkt entweder vereinzelt vorkommen darf oder jeweils ein Vielfaches darstellt (Abbildung 2.3). Daraus resultiert eine vereinfachte Ordnung von gängigen Rechnerarchitekturen. Beispiel hierfür ist ein Uniprozessor, welcher ausschließlich einen Befehlsstrom und einen Datenstrom nutzt (engl.: *single instruction stream, single data*, SISD). Moderne Rechnerarchitekturen hingegen fallen meist in mehrere Klassen, d.h. durch Vektorinstruktionen und Datenparallelismus in die Klasse der SIMD (engl.: *since instruction stream, multiple data*, SIMD), jedoch bedingt durch die Ausnutzung von multiplen Rechenkernen auch in MIMD (engl.: *multiple instruction stream, multiple data*, MIMD). MISD-Architekturen (engl.: *multiple instruction stream, single data*, MISD) hingegen sind rar und werden zumeist für Fehlererkennung genutzt.

Obgleich die Klassifizierung aus dem Bereich der Rechnerarchitekturen stammt, findet die Flynnsche Klassifizierung ebenfalls im Bereich der Software-Ausführung Anwendung. Jedoch wurde sie hierzu um SPMD (engl.: *single program, multiple data*, SPMD) und MPMD (engl.: *multiple program, multiple data*, MPMD) erweitert. Bedingt durch die schnelle und einfache Einordnung ist die Flynnsche Klassifizierung beliebt in der wissenschaftlichen Gemeinschaft, obgleich die hohe Abstraktion, die mangelnde Trennungsschärfe, als auch die beschränkte Merkmalserfassung prinzipiell gegen sie spricht.

2.1.5 Gesetz von Little

1961 führt Little den Beweis für die nach ihm benannte Warteschlangen-Gleichung (Abbildung 2.4) [Little, 1961]. Diese besagt, dass die durchschnittliche Menge an L Einheiten innerhalb eines Warteschlangensystems durch das Produkt der durchschnittlichen Ankunftsrate λ sowie der mittleren Verzögerung W pro Einheit definiert ist:

$$L = \lambda * W$$

Mehr als 30 Jahre später wandte Smith die Gleichung auf eine äußerst vereinfachte *Von-Neumann-Architektur* an, die von einem Einzelprozessorsystem ausgeht, dass Daten ohne Cachehierarchie direkt vom lokalen Hauptspeicher erhält [Smith, 1995]. Zugleich besteht ein Eins-zu-Eins-Verhältnis von jeder Fließkommaberechnung zu einer Lade-Instruktion. Smith leitet

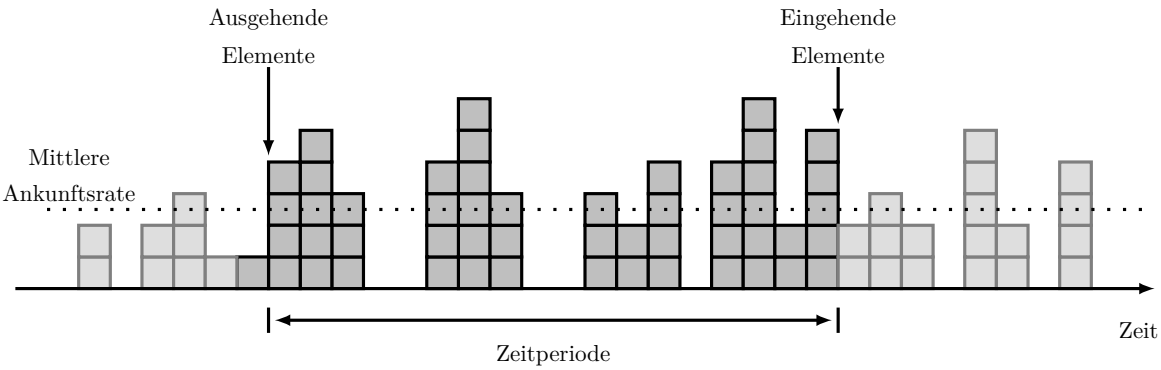


Abb. 2.4: Visualisierung des Gesetzes von Little.

daraus ab, dass in solch einem generalisierten System λ durch die Speicherbandbreite, W durch die Speicherlatenz und L durch die Nebenläufigkeit repräsentiert werden können:

$$Nebenläufigkeit = Bandbreite * Latenz$$

Da beide Seiten der Gleichung mit der Menge an Prozessoren multipliziert werden können, zeigt Bailey 1997 auf, dass die selbige Gleichung für verteilte Speichersysteme sogenannte *Shared-Memory*-Architekturen mit multiplen Prozessoren generalisiert werden kann [Bailey, 1997].

Es ist anzumerken, dass durch die Vernachlässigung einer Cachehierarchie, das Gesetz von Little mit den Anwendungen auf Rechnerarchitekturen von Smith und Bailey die untere Grenze der Nebenläufigkeit darstellt. Zugleich führen eine vergrößerte Bandbreite sowie Latenz zu einer erhöhten Menge an ausstehenden, d.h. in Bearbeitung befindlichen (engl.: *in-flight operations*) Ladeoperationen, wodurch für eine *Von-Neumann-Architektur* ohne Cachehierarchie eine erweiterte Lade-/Speicherwarteschlange benötigt wird.

2.1.6 Gesetz von Amdahl

Die theoretisch erzielbare Beschleunigung eines Programms, welches einen sequentiell und einen parallel ausführbaren Anteil beinhaltet, ist nach Amdahl primär durch den sequentiellen Anteil limitiert [Amdahl, 1967]. Hinzu kommt ein Sättigungseffekt der Parallelität, wodurch der Mehraufwand von Synchronisation und Datenaustausch ab einer gewissen Grenze der Parallelisierung zu geringeren Verarbeitungsgeschwindigkeiten führt und somit der sequentielle Programmteil dominieren muss.

Obgleich Amdahl seine Wahrnehmung selbst nicht formalisiert, kann eine Gleichung wie folgt hergeleitet werden. Gegeben sei die Beschleunigung Ψ, eine Problemstellung n, p Prozessoren, jeweiliger Laufzeit t, der grundsätzlich sequentielle Programmteil $\sigma(n)$, der parallele $\varphi(n)$ als auch der Mehraufwand zum Erzielen einer Parallelisierung $\kappa(n,p)$.

$$\Psi(n,p) = \frac{t_{sequentiel}(n)}{t_{parallel}(n,p)} = \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\frac{\varphi(n)}{p}+\kappa(n,p)}$$

Für große Problemstellungen kann der Mehraufwand vernachlässigt werden. Sei der grundsätzlich sequentielle Anteil s zusätzlich wie folgt beschrieben:

$$s = \frac{\sigma(n)}{\sigma(n) + \varphi(n)}, 0 \leq s \leq 1$$

Mit Umformung lässt sich daraufhin das Gesetz von Amdahl wie folgt formulieren:

$$\Psi(n, p) \leq \frac{1}{s + (1 - s)/p}$$

Die Formalisierung zeigt auf, dass die Problemstellung für Amdahl keine Auswirkung auf die Beschleunigung hat. Zugleich werden keine Prozessoreigenschaften in Betracht gezogen, sodass Verbesserung an Architektur oder Speicherhierarchie ebenso keine Auswirkungen haben.

2.1.7 Gesetz von Gustafson-Barsis

Gustafson erkennt bereits früh, dass nach dem Gesetz von Amdahl ein Programm immer durch dessen sequentiellen Programmteil limitiert ist, sodass selbst eine unendliche Anzahl an Prozessoren keine weitere Beschleunigung jenseits $\frac{1}{f}$ erzielt [Gustafson, 1988]. Zeitgleich zeigt sich jedoch durch ein Programm auf seinem Testsystem mit 1024 Prozessoren, dass dies entgegen der Erwartung dennoch der Fall ist. Daraus schlussfolgert er, dass die Vergrößerung des parallelen Anteils einer Problemstellung als auch eine größere Prozessoranzahl durchaus zu einer Beschleunigung führt, da im Umkehrschluss ein einzelner Prozessor diese Mehrarbeit sequentiell verrichten muss. Barsis formalisiert diesen Ansatz, ausgehend von Amdahls Beschreibung daraufhin wie folgt:

$$s = \frac{\sigma(n)}{\sigma(n) + \frac{\varphi(n)}{p}} \longrightarrow \sigma(n) = (\sigma(n) + \frac{\varphi(n)}{p}) \cdot s$$

$$1 - s = \frac{\frac{\varphi(n)}{p}}{\sigma(n) + \frac{\varphi(n)}{p}} \longrightarrow \varphi(n) = (\sigma(n) + \frac{\varphi(n)}{p}) \cdot (1 - s) \cdot p$$

Eingesetzt in $\Psi(n, p)$ und weiterem Auflösen ergibt sich:

$$\Psi(n, p) \leq p + (1 - p) \cdot s$$

Daraus ergibt sich, dass der sequentielle Programmanteil s weniger gewichtet wird als bei Amdahl, wodurch bei sehr kleinem s und einer großen Prozessoranzahl eine Beschleunigung zu erwarten ist.

2.1.8 Karp-Flatt-Metrik

Die beiden Gesetze nach Amdahl und Gustafson-Barsis stützen sich bei der Beschleunigungsberechnung eines Programms mit einem parallelen Programmanteil auf die Menge von Prozessoren sowie den grundsätzlich sequentiellen Anteil. Jedoch ist es schwierig den sequentiellen Anteil exakt zu ermitteln, sodass Karp und Flatt die tatsächlich gemessene parallele Beschleunigung mit Hilfe der Anzahl an Prozessoren nutzen, um Rückschlüsse über den sequentiellen Anteil zu ermitteln [Karp und Flatt, 1990].

Ausgehend vom Gesetz von Amdahl wird ein Kehrwert dessen gebildet und dieser weiter aufgelöst:

$$\frac{1}{\Psi(n,p)} \geq s + \frac{1}{p} - \frac{s}{p} \longrightarrow \frac{1}{\Psi(n,p)} + \frac{1}{p} \geq s(1 - \frac{1}{p})$$

Daraus und mit dem grundsätzlich sequentiellen Programmteil definiert als experimentell bestimmter serieller Programmteil e ergibt sich:

$$e \geq \frac{\frac{1}{\Psi(n,p)} - \frac{1}{p}}{1 - \frac{1}{p}}$$

Wird die Metrik auf eine Messreihe angewandt, bei welcher die Prozessoranzahl für eine gewisse Problemstellung variiert und zugleich die Zeit genommen wurde, so kann aus der jeweils erzielten Beschleunigung eine gegebenenfalls vorhandene Limitierung rückgeschlossen werden. Ist etwa für eine Messreihe der experimentell bestimmte serielle Anteil jeweils eng beieinander und gibt es keine weitere Beschleunigung ab einer gewissen Prozessoranzahl, so zeigt dies auf, dass der serielle Anteil klar dominiert und keine weitere Beschleunigung durch mehr Prozessoren zu erwarten ist. Variiert jedoch der experimentell ermittelte serielle Anteil in diesem Fall stark, so kann auf einen noch nicht adressierten Mehraufwand geschlossen werden.

2.1.9 Mooresches Gesetz

Im Jahr 1965 beschreibt Moore seine Beobachtung: Jede neue Generation eines Fertigungsprozess führt danach zu günstigeren Herstellungskosten von Transistoren und somit Komponenten [Moore, 1965]. Jedoch ist dieser Kostenpunkt nur erzielbar, wenn bei jeder neuen Technologie signifikant mehr Komponenten hergestellt und vertrieben werden. Zugleich zeigt er auf, dass sich die Transistordichte je integriertem Schaltkreis jährlich verdoppeln wird. Dies korrigiert Moore einige Jahre später, indem er von einer Verdopplung aller zwei Jahre spricht. Jedoch ist heutzutage das Mooresche Gesetz für eine Verdopplung der Anzahl an Transistoren pro Chip alle 18 Monate (d.h. $\approx 60\,\%$ mehr Transistoren je Chip pro Jahr) bekannt. Wissenschaft und Industrie leiteten aus dieser Technologieskalierung eine Geschwindigkeitsskalierung ab, sodass sich die Geschwindigkeit eines Prozessors ebenfalls alle 18 Monate verdoppeln sollte.

In den Jahren 2015 bis 2017 brach ein hitziger Streit in der wissenschaftlichen Gemeinschaft aus, ob das Mooresche Gesetz noch als korrekt anzusehen ist. Wissenschaftler wie Waldrop [Waldrop, 2016], Hennessy und Patterson [Hennessy und Patterson, 2011] (Abbildung 2.5) jedoch auch Intel mit der Ankündigung ihres neuen, dreiphasigen “Technologie, Architektur und Optimierung”-Schemas (im Gegensatz zum alten zweiphasigen “Tick-Tock”-Modell) zur Entwicklung neuer Prozessoren, zeigen auf, dass die 18- bis 24-monatige Regel zur Verdopplung der Transistoren pro Chip nicht mehr gültig sein kann. Andererseits ist gerade Intel daran interessiert das Gesetz am Leben zu halten, sodass im Jahr 2017 anhand des Begriffs *Hyper Scaling* die fortführende Existenz des Mooreschen Gesetzes aufgezeigt wurde. Danach werden nun einzelne Aspekte eines Transistors, d.h. *Fin-Pitch*, *Interconnect-Pitch*, *Gate-Pitch* und Transistor-Zellenhöhe im Detail betrachtet. Durch die Verkleinerung jedes einzelnen Aspekts und bedingt durch Intels im Gegensatz zur Konkurrenz ohnehin höhere Packdichte konnten die durchschnittlichen Kosten je Transistor weiter gesenkt werden und demzufolge dem Mooreschen Gesetz folgen [Renduchintala, 2017].

2.1.10 Die *Memory Wall*

Die technologische Grundlage der Leistungsfähigkeit von heutigen Rechnerarchitekturen sind die innerhalb von Prozessoren verwendeten, äußerst schnell getakteten Schaltkreise. Speicher nutzen hingegen kompakte sowie deutlich langsamere Schaltkreise, welche primär auf große Kapazitäten ausgelegt sind und damit vergleichsweise preiswert hergestellt werden können. Erst sekundär findet bei Speichern eine Optimierung hin auf Bandbreite und somit Geschwindigkeit statt.

Bereits 1995 beschreiben Wulf und McKee ein sich ständig zunehmendes Leistungsgefälle zwischen dem technologischen Fortschritt der Prozessoren auf der einen und den Speichern auf der anderen Seite [Wulf und McKee, 1995]. Zum Zeitpunkt ihrer Studie erkennen die Autoren

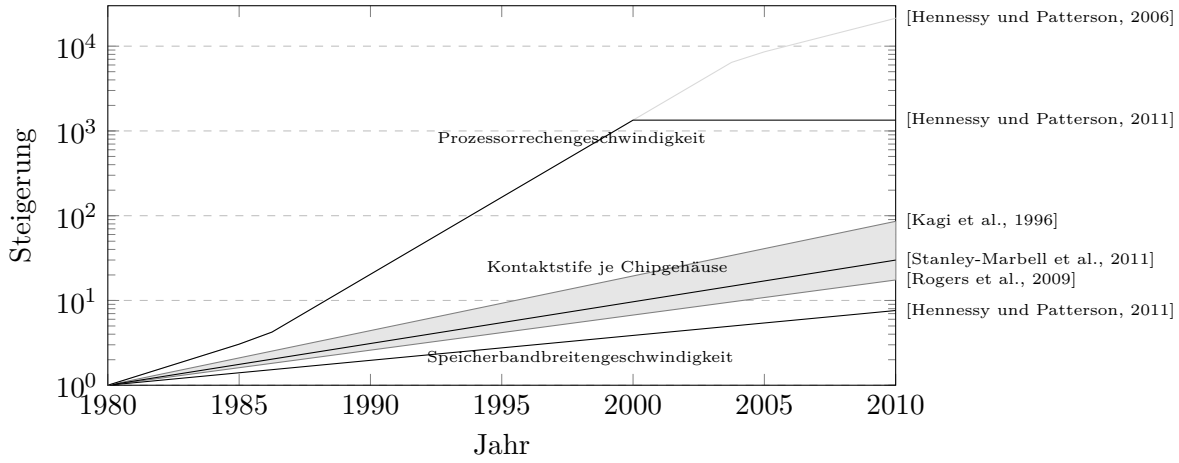


Abb. 2.5: **Die *Memory Wall*.** Hauptdetail stellt die Zunahme an Rechenleistung im Kontrast zur Speicherbandbreite in den Jahren 1980 bis 2010 dar [Hennessy und Patterson, 2006]. Zusätzlich ist die Verbesserung von Kontaktstiften (engl.: *pin*) je Chipgehäuse illustriert [Stanley-Marbell et al., 2011].

Wulf und McKee, dass beide Technologien in der Vergangenheit große Geschwindigkeitszuwächse erzielten, und gehen deshalb davon aus, dass dies sich auch in der Zukunft abbilden wird (Abbildung 2.5). Jedoch weisen die beiden darauf hin, dass der Leistungszuwachs bei Prozessoren um ein Vielfaches größer ausfällt und ausfallen wird als bei den Speicherarchitekturen.

Wulf und McKee veranschaulichen ihre Beobachtung anhand idealer Caches, welche eine erfolgreiche Trefferquote von über 99 % haben, während der technologische Geschwindigkeitsfortschritt für den Hauptspeicher als auch für den Prozessor mit dem gleichen Tempo voranschreitet. Dabei schlussfolgern sie, dass das Leistungsgefälle zwischen beiden sich innerhalb der nächsten fünf bis zehn Jahre dramatisch verschlimmern wird, sodass Wulf und McKee zu dem Ergebnis kommen, dass zukünftige Rechensysteme unmissverständlich durch den technologischen Fortschritt der Speichergeschwindigkeit dominiert werden. Daraus leiten Sie den Begriff der *Memory Wall* ab.

Da Wulf und McKee keine einfache Speichertechnologie erkennen konnten, welche linear zum Leistungszuwachs der Prozessortechnologie skalieren kann, entschieden Sie sich Aspekte zur Minimierung des Leistungsgefälles d.h. der *Memory Wall* aufzuzeigen. Der auffälligste sowie zu der Zeit wohl bedeutendste Aspekt für Softwareentwickler und Kompilerwerkzeuge ist die Notwendigkeit unterschiedliche Latenzen, d.h. die etwa durch einen *Non-Uniform Memory Access*

(NUMA) entstehen, als neuen *Status-Quo* für zukünftige Rechnersysteme anzuerkennen. In Anbetracht von heutigen dreidimensional gestapelten Speicherarchitekturen erringt dieser Aspekt eine signifikante Bedeutung, da diese im Gegensatz zu planaren Speichern multiple Stellen bieten, an welchen sich Latenzen unterschiedlich akkumulieren. Darunter fallen etwa der Zeilenspeicher (engl.: *row buffer*), der Kreuzschienenverteiler (engl.: *crossbar*), die Speicherbänke, Speicherbankverwalter, der interne Netzwerk-Multiplexer, der Verbindungsverwalter mit entsprechenden FIFOs, unterschiedliche Strategien, um Speicherseiten zu verwalten (z.B. *closed-page* oder *open-page policy*), sowie das Multiplexen von Adressen, welches wiederum zu unterschiedlichen Ressourcenkonflikten führen kann.

2.1.11 Die *Bandwidth Wall*

Wohingegen Wulf und McKee einzig auf die Symptome der sich weitenden Leistungslücke zwischen Prozessor- und Speichertechnologie hinweisen, untersuchen Burger et al. das zugrunde liegende Problem der externen Bandbreite, d.h. zwischen den Kontaktstiften (engl.: *pin*) der Prozessoreinheit und den Kontaktstiften der Speichereinheit, welches bereits von Elliott et al. als auch ein Jahr später von Kogge et al. (1996) herausgearbeitet wurde [Kagi et al., 1996]. Danach behaupten die Autoren, dass latenztolerante Techniken, welche Speicherzugriffslatenzen zum aktuellen Zeitpunkt der Studie minimieren, in der Zukunft die an den Kontaktstiften anliegende Bandbreitenlimitierung enthüllen werden. Daraus schlussfolgern Burger et al. eine fundamentale Hürde für Computersysteme, höhere Rechenleistung zu erreichen.

Obgleich Burger et al. eine Vielzahl an Aspekten zur Stützung dieser Hypothese nennt, sind die Wesentlichsten wie folgt: eine Prozessorarchitektur mit einer neuartigen Technologie wird mehr Ladeoperationen ausführen können und diese entsprechend ausstehend (*In-Flight*) haben. Dies führt zu einem deutlich erhöhten Druck auf das Speichersubsystem. Obwohl latenztolerante Techniken die erhöhte Anzahl an Ladeoperationen im besten Fall mithilfe von Lokalität entschärfen können, bieten diese dem Prozessor die Möglichkeit, Ladeoperationen schneller zu erstellen, womit sich die Bandbreitensituation zugleich verschlimmert. Werden mehr Daten und Instruktionen für einen weiteren Prozessor benötigt, so werden mehr Cachezeilen durch die Speicherhierarchie, d.h. vom Hauptspeicher zum Prozessor, transportiert. Jede Cachezeile hat das Potential einerseits zu wenig genutzt zu werden und andererseits wichtigere Cachezeilen zu verdrängen, wodurch ein signifikanter Mehraufwand in Datenübermittlung generiert wird. Entsprechend verlangt dies nach einer erhöhten Bandbreite. Hinzu kommen innerhalb einer Einheit multiple Rechenkerne und deren Hardware-Threads, welche die Bandbreitenproblematik weiter verschärfen.

Wenn auch die Bandbreite pro Kontaktstift stetig zunimmt, erwarten Burger et al. ein langsameres Wachstum für die Menge an Kontaktstiften (1996: 16 % pro Jahr [Kagi et al., 1996]; 2005: 10 % pro Jahr [Rogers et al., 2009]; 2011: eine Verdopplung von Kontaktstiften alle sechs Jahre, resultierend in $\sim$ 12 % pro Jahr [Stanley-Marbell et al., 2011]). Beides steht klar im Gegensatz zum massiven Zuwachs an Transistoren pro Chip (Verdopplung alle 18 Monate, d.h. $\sim$ 60 % pro Jahr), welcher von Moore vorhergesagt wurde [Moore, 1965]. Daraus resultiert, dass vornehmlich der Rechenleistungszuwachs zur so genannten *Bandwidth Wall* beiträgt, wohingegen sich Speicherbusse (Protokoll-Mehraufwand), Speicherschnittstellen, die Montageart, Leistungs- sowie Kühlanforderungen weniger auswirken (Abbildung 2.5) [Kagi et al., 1996, Rogers et al., 2009]. In Anbetracht des von Patterson et al. vorhergesagten Zuwachses an Speicherzugriffs-

zeit, welche sich im Bereich von 7 % pro Jahr bewegt [Hennessy und Patterson, 2011], wirkt die Zuwachsrate in der Gesamtzahl der Kontaktstifte nicht bedenklich. Jedoch hoben im Jahr

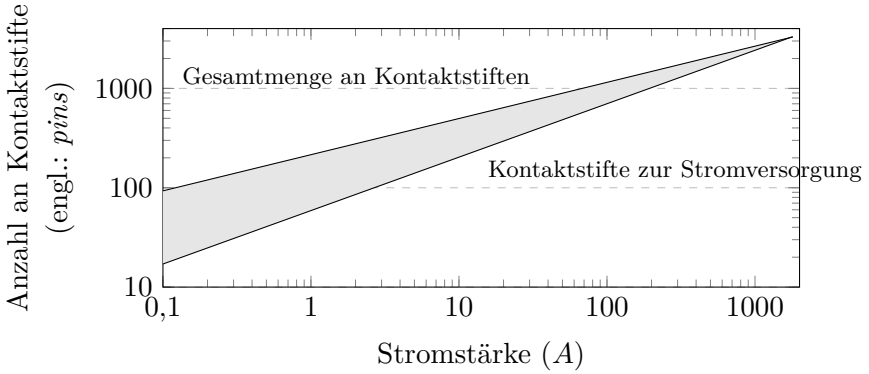


Abb. 2.6: Wachstum der Gesamtmenge an Kontaktstiften und Stromversorgungskontaktstiften in Relation zur Stromstärke. Dabei stellt sich eine Lücke an real nutzbaren Kontaktstiften ein [Stanley-Marbell et al., 2011].

2011 Stanley-Marbell et al. hervor, dass *Memory Wall* und *Power Wall* gleichermaßen durch die Skalierung von Kontaktstiften beeinträchtigt werden, da Stromversorgungskontaktstifte im Gegensatz zu tatsächlich nutzbaren Kontaktstiften ein deutlich stärkeres Wachstum aufweisen (Abbildung 2.6) [Stanley-Marbell et al., 2011]. Dies kann als Lücke an real nutzbaren Kontaktstiften (engl.: *Pin Wall*) angesehen werden.

Burger et al. sehen im Gegensatz zu Wulf und McKee einen Kompromiss zwischen Zugriffslatenz und Bandbreite vor, um die *Bandwidth Wall* zu entschärfen. Dies wird von Patterson unterstützt, welcher zugleich Bandbreite bevorzugt, da Prozessorarchitekten potentiell mehr Rechenleistung aus Bandbreite statt Latenz extrahieren können [Patterson, 2004].

Sofern davon auszugehen ist, dass die Rechenleistung von neuartigen Prozessorarchitekturen nicht weiter dem Mooreschen Gesetz folgen wird (Unterabschnitt 2.1.9), würde dies implizit zu einer potentiellen Linderung der *Bandwidth Wall* zutragen. Zugleich wäre aus der Sicht eines Einzelprozessors ebenfalls eine implizite Entlastung hinsichtlich der notwendigen Zunahme an Kontaktstiften vorhanden. Im Angesicht dessen, dass multiple Prozessoreinheiten an einem gemeinsamen Speicher betrieben werden können, wird die Speicherbandbreite jedoch selbst nach dem Ende des Mooreschen Gesetzes als Hemmnis für höhere Rechenleistung bestehen bleiben, weshalb neuartige Konzepte zur Steigerung benötigt werden [Shalf und Leland, 2015].

2.1.12 Die *Power Wall* und die *Energy Wall*

Aufgrund des Minimierungsgrads in der CMOS-Technologie, der gleichzeitigen Zunahme der Transistordichte, sowie der Erhöhung der Taktfrequenz waren Prozessorarchitekturen zunehmend der *Power Wall* ausgesetzt, da die Verlustleistung (Watt pro cm^2) aufgrund von höheren Leckströmen dramatisch zunahm [Pollack, 1999]. Bedingt durch die Grenzen in der Thermik, d.h. Kühlung, hatte dies das Potential, zukünftige Prozessoren zum Schmelzen zu bringen.

Der letzte Prozessor, welcher die energetischen und thermischen Beschränkungen vernachlässigte, war der Intel Pentium 4 mit NetBurst-Architektur, der für eine hohe Rechenleistung eine äußerst tiefe Verarbeitungs-Pipeline mit hohen Taktfrequenzen nutzte. Als Konsequenz müssen

heutige Rechnerarchitekturen neue Richtungen einzuschlagen, um zugleich die *ILP Wall* zu vermeiden. Dies führte zur Entwicklung von neuen Technologien im Bereich der Multi- als auch der Vielkernprozessoren, welche Parallelität und Nebenläufigkeit zur weiteren Leistungssteigerung ausnutzen [Bose, 2012]. Jedoch verschärft das einfache Zusammenstellen von mehreren Prozessorkernen ebenfalls Energiebedarf, Produktionskosten und Die-Fläche. Daher geht der aktuelle Trend in die Richtung Heterogenität.

Wohingegen Prozessorarchitekturen meist direkt mit der *Power Wall* assoziiert werden, tragen Speicher gleichermaßen dazu bei [Bose, 2012]. Insbesondere das wiederholte Ein- und Auslesen zum Auffrischen des Speicherinhalts aufgrund von Leckströmen, die Puffer-Chips, die Serialisierer/Deserialisierer-Verbindungen sowie das Auslesen von Datenblöcken, welche nur zu Teilen genutzt werden, tragen maßgeblich zum Leistungsbudget bei und verschärfen dadurch die *Power Wall*.

Innerhalb des Kontext ExaFLOPS wird die sogenannte *Energy Wall* oft als Synonym für die *Power Wall* genutzt. Jedoch ist die *Energy Wall* im Gegensatz zur *Power Wall* keine technologische und somit physikalische Beschränkung, sondern eine von der DARPA eingeführte Zielsetzung, welche in einem Bericht aus dem Jahr 2008 für einen entsprechenden Exascale-Höchstleistungsrechner erwähnt wird [Bergman et al., 2008]. Laut diesem DARPA-Bericht, erstellt von Kogge et al., soll bis zum Jahr 2020 ein Exascale-Höchstleistungsrechner errichtet werden, welcher das maximale Gesamtleistungsbudget von 20 Megawatt[1] nicht überschreitet. Dies führt zum Ergebnis, dass zukünftige Rechnerarchitekturen maximal 50 Picojoule pro Fließkommaberechnung nutzen dürfen, was aktuelle Angebote nicht annähernd erfüllen können (aktuell: 190 $\frac{\text{pJ}}{\text{FLOP}}$, vorhersehbare Zukunft: zwischen 180 und 420 $\frac{\text{pJ}}{\text{FLOP}}$ [Kogge, 2015]. Mit Blick auf den Trend weg von leichtgewichtigen Systemen wie dem IBM BlueGene/Q hin zu schwergewichtigen, heterogenen Höchstleistungsrechnern wie dem Cray Titan, verlangt diese Divergenz drastische Innovationen, denn selbst eine vereinfachte Extrapolation von aktuell leichtgewichtigen Architekturen benötigt als Minimum 2 Gigawatt für Exascale [Bose, 2012].

Aus den Effekten der *Memory Wall*, *Power Wall* und der *ILP Wall* leitet schlussendlich Patterson eine *Brick Wall* ab, welche nach ihm als neue, "gebräuchliche Weisheit für Rechnerarchitekturen" anzusehen ist [Patterson, 2006].

2.1.13 Die *Locality Wall*

Moderne Hochleistungsrechner welche mit mehreren Prozessorsockeln und heterogen ausgeführten Beschleunigern bestückt sind, besitzen uneinheitlich ausgeführte Speichersubsysteme. Auf der einen Seite existiert der reguläre Hauptspeicher, welcher bei einem Multiprozessorsystem in Gruppen an die Sockel angeschlossen ist. Entsprechend müssen nichtlokale Dereferenzierungen an einen entfernten Prozessor delegiert werden, jedoch ist die Verschaltung zwischen den Sockeln bezüglich der Bandbreite meist langsamer ausgeführt als die Bandbreite zwischen Sockel und Speicher. Hinzu kommt, dass Beschleuniger über einen dedizierten Bus (z.B. PCIe oder NVlink) angeschlossen sind und zugleich eigenen Speicher bereitstellen. Dieser ist häufig als *cache-coherent NUMA* (ccNUMA) ausgeführt (Abschnitt 3.2), sodass Prozessoren und Beschleuniger in beide Speicher kohärent speichern und auch davon lesen können. Jedoch ist die

[1] Das größte US-Forschungslabor *Lawrence Livermoore National Laboratory* kann maximal 20 Megawatt in die Halle des Höchstleistungsrechners einspeisen.

Bandbreite auf dem Bus deutlich geringer als zwischen Beschleuniger und dediziertem Speicher [Ding, 2017].

Es resultiert, dass Speicherdereferenzierungen auf nichtlokale Speicher äußerst geringe Speicherbandbreiten erzielen. Anhand verschiedener Strömungsbenchmarks, datenintensiver Anwendungen sowie insbesondere per konjugierten Gradientenbenchmark HPCG zeigt Kogge im Jahr 2017 auf, dass eine gewisse *Locality Wall* existieren muss [Kogge, 2017]. Danach besteht die Hauptschwierigkeit darin, irreguläre Zugriffe einer Anwendung zu vermeiden und eingeholte Cachezeilen so häufig wie möglich wiederzuverwenden. Kogge zeigt dabei einen Ausweg auf, indem die Arbeit, d.h. ein Hardware-Thread an die Stelle migriert wird, an welcher die Daten gespeichert sind. Dies steht im Gegensatz zum bewährten Verfahren, welche die Daten an die Stelle zu bewegen, an welcher gerade die Arbeit ausgeführt wird.

2.1.14 Roofline-Modelle

Ein *Roofline*-Modell (deutsch: *Dachlinien*-Modell) ist ein einfaches, häufig genutztes und sehr grobes Visualisierungsinstrument, um die Charakteristik einer Rechnerarchitektur ins Verhältnis zur arithmetischen Intensität (AI) von Algorithmen zu setzen [Williams et al., 2009]. Die arithmetische Intensität eines Algorithmus wird dabei als Relation der Anzahl an Fließkommaberechnungen zur Menge an zu ladenden und speichernden Daten in Bytes definiert [Ofenbeck et al., 2014]. Dabei gilt:

$$\text{max. GFLOP/s} = \min \begin{cases} \text{max. GFLOP/s} \\ \text{stream GB/s} \cdot \dfrac{\text{FLOP}}{\text{Byte}} \end{cases}$$

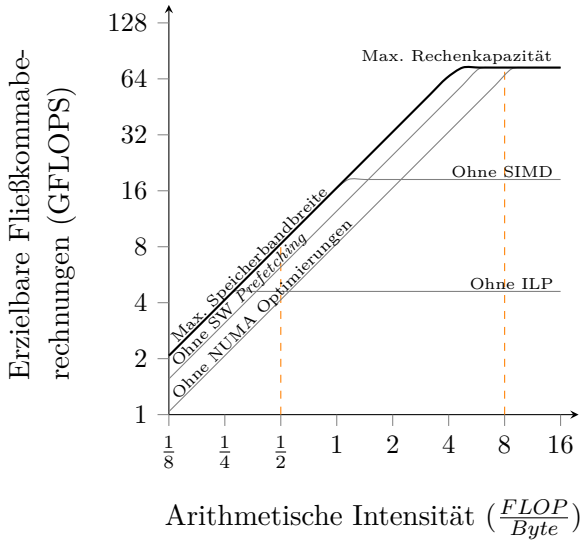


Abb. 2.7: *Roofline*-Modell eines Prozessors vom Typ AMD Opteron 2356 (Barcelona). Fiktiv wurden zwei arithmetische Intensitäten eingetragen, wovon die eine mit $\frac{1}{2}$ durch die Speicherbandbreite, die andere mit 8 durch die Rechenkapazität beschränkt ist.

Im Detail können nun arithmetische Intensitäten von unterschiedlichen Algorithmen auf der x-Achse aufgetragen werden. Wird die arithmetische Funktion der gewählten Architektur

vertikal tangiert, zeigt sich, ob eine Beschränkung durch die Speicherbandbreite oder aber der Rechenkapazität vorliegt (Abbildung 2.7).

Durch die Einfachheit können *Roofline*-Modelle ausschließlich idealisierte Algorithmen betrachten, bei welchen Kommunikation und Rechnerarchitektureffekte wie NUMA-Zugriffe nebensächlich sind. Mithilfe dieser einfachen Visualisierung kann erkannt werden, ob ein Algorithmus beschränkt durch die Speicherbandbreite oder durch die Rechenkapazität ist, sodass gerade im ersten Fall eine Optimierung mit Hilfe von dedizierten Rechnerarchitektureigenschaften wie etwa eine Vektorisierung oder aber Software-Threading zur Verwendung paralleler arithmetisch, logischer Einheiten größtenteils unnötig ist. Unter Umständen kann jedoch mit Softwaretechniken wie das Ausrollen von Schleifen, einer Quelltextumstrukturierung oder gar lang laufenden Schleifen die Speicherbandbreitenlimitierung minimiert werden. Oder gerade entgegengesetzt, d.h. dass ein Algorithmus durch Rechenkapazität limitiert wird, sodass mehr Speicherbandbreite keinen, jedoch die explizite Ausnutzung der genannten Techniken entsprechend Einfluss haben würde. Wurde bereits eine Optimierung eines Algorithmus vollzogen, kann ein *Roofline*-Modell auch im Nachhinein grob validieren, ob weiteres Optimierungspotential vorhanden ist.

Größtes Problem von *Roofline*-Modellen ist die Einfachheit, welche gleichzeitig dessen Stärke ist. Dies hat zur Folge, dass heutige Problemstellungen wie Energieeffizienz und Zuverlässigkeit von Systemen nicht betrachtet werden können, andererseits ein schnelle Evaluation der Beschränktheit erstellt werden kann. Zudem betrachtet ein *Roofline*-Modell jeweils nur eine Rechnerarchitektur, sodass Effekte aus heterogenen Systemen und deren zwingend nötigen Kommunikationsmuster nicht in Korrelation gebracht werden können, um etwa Rückschlüsse auf höhere Recheneffizienz zu ziehen.

2.1.15 Schwache und starke Skalierung

Schwache (engl.: *weak scaling*) und starke Skalierung (engl.: *strong scaling*) sind zwei Terminologien, welche im Bereich des Höchstleistungsrechnens (engl.: *high performance computing*, HPC; *supercomputing*) genutzt werden. Sie geben an, inwiefern ein Algorithmus eine gewisse Menge an Prozessoren nutzen kann, um entsprechend hohe Rechenleistung erzielen zu können. Bei beiden wird die Anzahl der zur Verfügung gestellten Prozessoren variiert, jedoch wird bei der starken Skalierung die Gesamtproblemgröße einer Anwendung festgelegt. Im Kontrast dazu wird bei der schwachen Skalierung die Problemgröße je Prozessor festgesetzt. Schwache Skalierung ist somit leichter zu erreichen, da für solch Szenario ohnehin eine hochgradig parallele Anwendung benötigt wird, sodass ausschließlich mehr Daten für mehr Prozessoren zur Verfügung gestellt werden müssen. Im Gegensatz dazu ist die starke Skalierung meist nicht trivial zu erzielen, da Anwendungen nach Amdahl (Unterabschnitt 2.1.6) einen sequentiellen Anteil besitzen, der nicht weiter aufgeteilt werden kann. Somit kann nur der parallele Anteil jeweils weiter aufgeteilt werden, wodurch bei einer zunehmenden Anzahl an Prozessoren der sequentielle Anteil vermehrt dominiert und zumeist der Datenaustausch unter den Prozessoren gleichzeitig zunimmt.

2.2 Seismische Berechnungsverfahren

Einen integralen Bestandteil der modernen Geologie stellen seismische Berechnungsverfahren dar. Diese ermöglichen es, unerforschten und auch besonders tiefen Untergrund ohne Aushe-

bung, sowie zum Teil ohne Beeinflussung dessen, mit algorithmischer Hilfe sichtbar zu machen. Unterschiedlich schnelle Schallwellenausbreitung resultiert in einem zweidimensionalen Bild oder dreidimensionalen Volumen, welches Auskunft gibt über die sich im Untergrund befindlichen Gesteinsformationen, entsprechend Materialien. Trotz signifikant hoher Rechenanforderung ist dies insbesondere für den Öl- und Gasbereich von Interesse, da Testbohrungen nur noch an den Orten vorgenommen werden müssen, an welchen ein gewisser Erfolg vorab berechnet worden ist.

Innerhalb der seismischen Berechnungsverfahren existieren verschiedene Varianten, wobei Kirchhoff-Migration, Reverse Time Migration (RTM) sowie Full-Waveform Inversion (FWI) die bekanntesten darstellen. Diese wiederum können als isotropisch, vertikal querlaufend isotropisch (engl.: *vertical transverse isotropic*, VTI) oder gar schräg querlaufend isotropisch (engl.: *tilted transverse isotropic*, TTI) betrieben werden. War die Kirchhoff-Migration bis vor einigen Jahren das wichtigste seismische Berechnungsverfahren, ist aktuell die Reverse Time Migration bedingt durch bessere Ergebnisse von höchstem Interesse (Abbildung 1.1). Full-Waveform Inversion stellt die Zukunft der seismischen Berechnungsverfahren dar, hat jedoch äußerst hohe Anforderungen an die Rechenleistungen. Entsprechend wird im professionellen Bereich ein Mix aus Kirchhoff-Migration, Reverse Time Migration und Full-Waveform Inversion eingesetzt, wobei Reverse Time Migration die Hauptlast zufällt.

Der nachfolgende Abschnitt wird sich deshalb mit der Reverse Time Migration beschäftigen, welche aus dem Entwicklungssatz von Taylor und der Finite-Differenzen-Methode konstruiert werden kann. Zusätzlich wird das *Ricker Wavelet* aufgezeigt, welches häufig für synthetische Anwendungen als seismischer Puls Anwendung findet. Zu guter Letzt wird ein Einblick in seismische Werkzeugsammlungen und Demodaten gegeben.

2.2.1 Der Entwicklungssatz von Taylor

Mit einer Tangente, d.h. einem Polynom ersten Grades, kann eine differenzierbare Funktion $f(x)$ in einem lokalen Entwicklungspunkt c durch $t(x) = f(c) + f'(c)(x-c)$ linear approximiert werden. Soll jedoch eine Approximation einer n-fach differenzierbaren Funktion f mit einer Ableitung höherer Ordnung erstellt werden, so verhilft der Entwicklungssatz von Taylor, oft auch als Taylorentwicklung tituliert, durch ein Schmiegpolynom n-ten Grades zu einer besseren Annäherung. Dabei gilt folgendes (Theorem sowie Definition aus [Wolff et al., 2004]):

Theorem 2.2.1 (Entwicklungssatz von Taylor). *Sei J ein Intervall und $f : J \to \mathbb{R}$ sei eine $(n+1)$ mal differenzierbare Funktion auf J. Sei $c \in J$ beliebig. Dann gibt es zu jedem $x \in J$ eine Zwischenstelle y zwischen x und c mit*

$$\begin{aligned} f(x) &= f(c) + \frac{f'(c)}{1!}(x-c) + \frac{f''(c)}{2!}(x-c)^2 + \cdots + \frac{f^{(n)}(c)}{n!}(x-c)^n + \frac{f^{(n+1)}(y)}{(n+1)!}(x-c)^{n+1} \\ &= \sum_{k=0}^{n} \frac{f^{(k)}(c)}{k!}(x-c)^k + \frac{f^{(n+1)}(y)}{(n+1)!}(x-c)^{n+1}. \end{aligned}$$

Definition 2.2.2 (Taylorpolynom). *Das Polynom $\sum_{k=0}^{n} \frac{f^{(k)}(c)}{k!}(x-c)^k$ heißt Taylorpolynom n-ten Grades um den Entwicklungspunkt c, der Term $\frac{f^{(n+1)}(y)}{(n+1)!}(x-c)^{n+1}$ Restglied.*

Entsprechender Beweis wird in Appendix A.1 aufgezeigt.

2.2.2 Der Entwicklungssatz von Taylor in der n-ten Dimension

Der in Unterabschnitt 2.2.1 eingeführte Entwicklungssatz von Taylor kann generalisiert und somit für die n-te Dimension ($\mathbb{R}^n$) angegeben werden. Hierfür wird die partielle Ableitung benötigt, welche wie folgt definiert ist (Definition aus [Wolff et al., 2004]):

Definition 2.2.3 (Partielle Ableitung). *Sei $U \subseteq \mathbb{R}^p$ offen, $f : U \to \mathbb{R}^q$ sei eine Abbildung und $x \in U$ sei fest gewählt. Sei $e_j = (0, 0, \ldots, 0, 1, 0, \ldots, 0)^t$, wo die 1 an der j-ten Stelle steht, der j-te kanonische Basisvektor. f heißt partiell nach der Koordinate x_j differenzierbar, wenn der Grenzwert*

$$f_{x_j}(x) = \lim_{0 \neq h \to 0} \frac{1}{h}(f(x + he_j) - f(x))$$

existiert. $f_{x_j}(x)$ heißt partielle Ableitung nach x_j. Sie wird auch mit $\frac{\partial f(x)}{\partial x_j}$ bezeichnet.

Zur einfacheren Schreibweise des Taylorschen Entwicklungssatz in $\mathbb{R}^n$, wird zumeist eine Multiindex-Notation eingeführt (Definition aus [Rannacher, 2010, Walter, 2007]):

Definition 2.2.4 (Multiindex-Notation). *Ein n-dimensionaler "Multiindex" ist ein n-Tupel $\alpha = (\alpha_1, \ldots, \alpha_n)$ mit nicht-negativen ganzzahligen Komponenten α_i ($\in \mathbb{N}_0$). Für Multiindizes sind eine "Ordnung" $|\alpha|$ und die "Fakultät" $\alpha!$ definiert durch*

$$\begin{aligned} |\alpha| &:= \alpha_i + \cdots + \alpha_n \\ \alpha! &:= \alpha_1! + \cdots + \alpha_n! \\ \vec{h}^{\,\alpha} &:= h_1^{\alpha_1} \ldots h_n^{\alpha_n}, h = (h_1, \ldots, h_n) \in \mathbb{R}^n \end{aligned}$$

Für eine $|\alpha|$-mal stetig differenzierbare Funktion wird gesetzt:

$$D^\alpha f := \partial_1^{\alpha_1} \cdot \cdots \cdot \partial_n^{\alpha_n} f = \frac{\partial^{|\alpha|} f}{\partial x_1^{\alpha_1} \cdot \cdots \cdot \partial x_n^{\alpha_n}}.$$

Wegen der Stetigkeit der Ableitungen ist dieser Ausdruck unabhängig von der Reihenfolge der partiellen Ableitungen.

Mit Hilfe beider vorausgehenden Definitionen, kann die multidimensionale Taylorentwicklungsformel wie folgt definiert werden (Definition aus [Walter, 2007, Rannacher, 2010]):

Theorem 2.2.5 (Entwicklungssatz von Taylor in $\mathbb{R}^n$). *Sei $U \subset \mathbb{R}^n$ eine offene Menge, $\vec{c} \in U$ und $f : U \to \mathbb{R}$ eine $(r+1)$-mal stetig differenzierbare Funktion. Dann gibt es für jedes $\vec{x} \in U$, für welches die Strecke zwischen $\vec{c}$ und $\vec{x}$ ganz in U liegt, eine "Zwischenstelle" ξ auf dieser Strecke, d.h. ein $\xi = \vec{c} + t(\vec{x} - \vec{c})$ für ein geeignetes $0 \leq t \leq 1$ mit*

$$f(\vec{x}) = \sum_{|\alpha| \leq r} \frac{D^\alpha f(\vec{c})}{\alpha!}(\vec{x} - \vec{c})^\alpha + \sum_{|\alpha| = r+1} \frac{D^\alpha f(\xi)}{\alpha!}(\vec{x} - \vec{c})^\alpha.$$

Für den vollständigen, länglichen Beweis wird auf [Walter, 2007] und [Rannacher, 2010] verwiesen. Im Detail wird die eindimensionale Taylorentwicklung mehrfach angewandt, um den Beweis daraufhin per Induktion und mit Hilfe der Kettenregel zu vollenden.

2.2.3 Finite-Differenzen-Methoden (FDM)

Finite-Differenzen-Methoden werden in der Numerik genutzt, um gewöhnliche und partielle Differentialgleichungen zu approximieren, also näherungsweise zu berechnen.

Beispielhaft soll hierfür jeweils der linke sowie rechte einseitige und der daraus entwickelbare zentrale Differenzenquotient bestimmt werden. Mit Hilfe der Taylorentwicklung (Unterabschnitt 2.2.1) soll der rechte einseitige Differenzenquotient an dem Entwicklungspunkt $x = c+h$ berechnet werden. Dabei gilt (R: Restglied):

$$\begin{aligned} f(x) &= f(c) + \frac{f'(c)}{1!}(x-c) + \cdots + R_n(x) \\ f(c+h) &= f(c+h-h) + \frac{f'(c)}{1!}(c+h-c) + \cdots + R_n(x) \\ f(c+h) &= f(c) + h \cdot f'(c) + \cdots + R_n(x) \\ f(c+h) &= f(c) - h \cdot f'(c) + R_1(x) \\ f'(c) &\approx \frac{f(c+h) - f(c)}{h}, \lim_{h\to 0} \frac{R_1(x)}{h} = 0 \end{aligned}$$

Der linke einseitige Differenzenquotient erster Ordnung lässt sich auf die gleiche Weise mit $c = x - h$ entwickeln, wodurch folgendes gilt:

$$f'(c) \approx \frac{f(c) - f(c-h)}{h}$$

Der linke als auch rechte einseitige Differenzenquotient erfüllen bedingt durch $R_1(x) = \mathcal{O}(h)$ die erste Ordnung. Hingegen bildet der zentrale bzw. symmetrische Differenzenquotient eine zweite Ordnung:

$$\begin{aligned} f'(c) &= \frac{1}{2}\left(\frac{f(c) - f(c-h)}{h} + \frac{f(c+h) - f(c)}{h}\right) \\ f'(c) &\approx \frac{f(c+h) - f(c-h)}{2h} = f'(c) + \mathcal{O}(h^2) \end{aligned}$$

Für eine Extrapolation von Differentialgleichungen höherer Ordnung kann der Algorithmus der Romberg-Integration angewandt werden, sofern das numerische Verfahren, allgemein beschrieben mit $T(h)$, die Bedingung $\lim_{h\to 0} T(h) = \tau_0$ erfüllt, wobei τ_0 bereits die Lösung des Verfahrens ist.

2.2.4 Akustisch homogene Wellengleichung

Zur Bildung der akustisch homogenen Wellengleichung wird der sogenannte Laplace-Operator benötigt, welcher wie folgt definiert ist (Definition aus [Weickert, 2004]):

Definition 2.2.6 (Laplace-Operator). *Sei $U \subset \mathbb{R}^n$ eine offene Menge und f in $c \in U$ zweimal partiell differenzierbar. Dann wird*

$$\Delta f(c) = \sum_{k=1}^{n} \frac{\partial^2 f}{\partial x_k^2}(c)$$

der Laplace-Operator von f in c genannt.

Satz 2.2.7. *Sei die Schallgeschwindigkeit $c \in \mathbb{R}^{>0}$ und der Schalldruck $p \in \mathbb{R}$. Sei zusätzlich $p(t, x_1, \ldots, x_n)$ zweimal partiell differenzierbar, so gilt für die akustisch homogene Wellengleichung bei kleinen Störungen und ruhendem Medium (Beweis [Wimmer-Schweingruber, 2007]):*

$$\Delta p - \frac{1}{c^2} \frac{\partial^2 p}{\partial t^2} = 0.$$

Bedingt durch die aus der Taylorentwicklung entstandene Approximation ist die akustisch homogene Wellengleichung nicht stabil und führt durch stetiges Einwirken eines Annäherungsfehlers zur Dispersion. Eine numerisch stabile Lösung für n Dimensionen, der Geschwindigkeit u sowie diskretem Zeitschritt Δt und diskreter Länge Δh stellt die nach Courant, Friedrichs und Lewy benannte CFL-Zahl c dar [Lewy et al., 1928]:

$$c = \Delta t \sum_{i=1}^{n} \frac{u_{h_i}}{\Delta h_i} \leq c_{max}.$$

Nach Sei kann für homogene Wellengleichungen die CFL-Zahl z.B. bei zweifacher Ordnung als $c \leq \frac{1}{\sqrt{2}}$ und bei vierfacher Ordnung als $c \leq 0,606$ gesetzt werden [Sei, 1995]. Jedoch führt die CFL-Zahl nur zur Verminderung der numerischen Dispersion, weshalb Thorbecke aufzeigt, dass für die diskrete Länge Δh in Relation zur CFL-Zahl c und Frequenz f

$$\Delta h < \frac{c_{min}}{5 \cdot f_{max}}$$

gelten sollte, sodass numerisch stabile Ergebnisse erzielbar werden [Thorbecke, 2017].

2.2.5 Reverse Time Migration (RTM)

Die im Jahre 1989 von Etgen erbrachte Studie zur akkuraten Modellierung der Wellengleichung (engl.: *acurate wave equation modeling*) illustriert die historische Wurzeln [Etgen und Dellinger, 1989]. Danach wurde eine Grundlage von Kelly et al. (1976) und zusätzlich von Marfurt (1984) zur Modellierung erbracht, indem zweidimensionale, elastische Wellengleichungen mit Hilfe von Finite-Differenzen-Methoden der ersten sowie zweiten Ordnung dargestellt wurden [Kelly et al., 1976, Marfurt, 1984]. Aufbauend auf FDM sowie den Vorarbeiten von McMechan (1982) [McMechan, 1982] und Whitmore (1983) [Whitmore, 1983] stellt Baysal et al. (1983) die Reverse Time Migration (RTM) vor, welche bis zum aktuellen Zeitpunkt als de-facto Standard für seismische Wellenausbreitung genutzt wird [Baysal et al., 1983].

Allgemein wird unter der seismischen Reverse Time Migration (RTM) eine Rückwärts-Extrapolation von seismischen Ausgangsdaten verstanden, welche an einem Zeitpunkt t_{max} vorliegen. Bei jedem Zeitschritt in Richtung t_0 wird im zweidimensionalen Fall über die x- und z-Richtungen, bzw. im dreidimensionalen ebenfalls in y-Richtung, eine Extrapolation getätigt. Dadurch entsteht mit jedem weiteren Zeitpunkt eine Rückwärtsbewegung und somit eine Migration von seismischen Wellen, wodurch die seismische Eingangsdaten rekonstruiert werden können. Im Falle von RTM wird auf die bidirektionale, akustisch homogene Wellengleichung zurückgegriffen (Unterabschnitt 2.2.4). Innerhalb von synthetischen Anwendungen findet zumeist nicht nur eine rückwärtsgerichtete, sondern zugleich eine vorwärtsgerichtete Migration statt, welche im Nachgang jeweils überlagert werden können (Abbildung 2.8). Das Resultat wird zur Validierung der Eingangs- sowie der Ausgangsdaten genutzt.

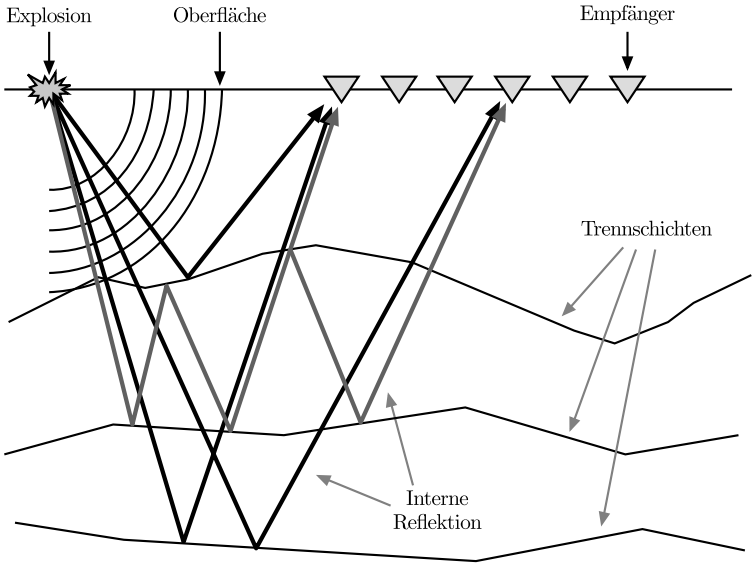


Abb. 2.8: Beispielhafte Ausbreitung von seismisch akustischen Wellen. Die Migration in Richtung der Empfänger wird als vorwärtsgerichtet bezeichnet, die Migration in Richtung der Explosion als rückwärtsgerichtet.

Im Detail nutzt die zeitliche Ableitung in t der Reverse Time Migration eine zentrale Finite-Differenzen-Methode zweiter Ableitung, erster Ordnung (Unterabschnitt 2.2.3), sodass für eine n-dimensionale, rückwärtsgerichtete Reverse Time Migration mit diskretem Zeitintervall Δt und Laplace-Operator auf p gilt (Herleitung Abschnitt A.2):

$$p(x_1, \ldots, x_n, t - \Delta t) = 2p(x_1, \ldots, x_n, t) - p(x_1, \ldots, x_n, t + \Delta t) + c^2 \Delta t^2 \Delta p$$

bzw.

$$p(x_1, \ldots, x_n, t - \Delta t) = 2p(x_1, \ldots, x_n, t) - p(x_1, \ldots, x_n, t + \Delta t) + c^2 \Delta t^2 \sum_{k=1}^{n} \frac{\partial^2 p}{\partial x_k^2}$$

Hingegen gilt für die n-dimensional vorwärtsgerichtete Reverse Time Migration:

$$p(x_1, \ldots, x_n, t + \Delta t) = 2p(x_1, \ldots, x_n, t) - p(x_1, \ldots, x_n, t - \Delta t) + c^2 \Delta t^2 \Delta p$$

2.2.6 Ricker Wavelet

Für synthetisch seismische Anwendungen wird häufig ein Puls benötigt, welcher durch stetiges Einwirken in ein zweidimensionales Feld oder einen dreidimensionalen Raum eine seismische Welle erzeugt. Eines solcher synthetischen Initialwellen stellt das nach dem amerikanischen Geophysiker Ricker benannt sogenannte *Ricker Wavelet* dar (Abbildung 2.9). Danach berechnet Rickers Gleichung anhand der maximalen Frequenz f_{max} sowie eines bestimmten Zeitpunktes t die maximale Amplitude relativ zur maximalen Frequenz wie folgt:

$$f(t) = (1 - 2\pi^2 f_{max}^2 t^2) e^{-\pi^2 f_{max}^2 t^2}$$

Bedingt durch die Exponentialfunktion sowie Potenzen ist das *Ricker Wavelet* äußerst rechenaufwändig. Sofern jedoch die Zeitpunkte bekannt sind, können diese vorab berechnet und in Datenfeldern vorgehalten werden. Eine Alternative zum *Ricker Wavelet* stellt die klassische, gaußsche Glocke dar.

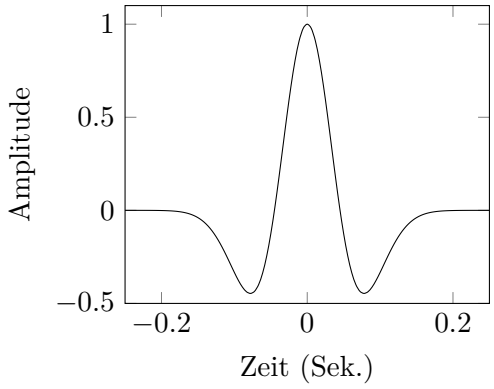


Abb. 2.9: Das *Ricker Wavelet*. Die Amplitude zeigt die relative, maximale Frequenz, welche für dieses Beispiel auf 5 Hz gesetzt wurde.

2.2.7 Seismische Werkzeugsammlung und Demodaten

Mit Unterstützung namhafter Öl- und Gasfirmen vertreibt die Fakultät *Center for Wave Phenomena* (CWP) der Technischen Universität Colorada die beiden frei verfügbaren seismischen Werkzeugsammlungen *CWP Seismic Un*x* (SU) und *Madagascar* [Stockwell, 2011]. *Madagascar* baut hierbei auf SU auf. Ursprünglich als geophysikalisches Ausbildungswerkzeug entwickelt, erfreut sich SU einer großen Popularität, sodass dessen RSF-Datenformat zum Speichern von seismischen Datensätzen neben dem industriellen Datenformat SEG-Y als Standard angesehen werden kann. SU kapselt seismische, analytische und bilderzeugende Algorithmen in einzelne, eigenständige C-Programme, welche mit Hilfe von *Bash*-Skripten zu Sequenzen aus Verarbeitungsschritten zusammengesetzt werden können. Der Austausch zwischen den einzelnen Programmen basiert entsprechend auf dem RSF-Datenformat, wobei auch SUs internes Datenformat Anwendung findet, welches ohnehin in RSF eingebettet ist. Zwei der bilderzeugenden C-Programme aus SU sind *Ximage* sowie *psimage*, welche innerhalb dieser Arbeit Verwendung finden.

SU bietet eine solide Basis für seismische Berechnungsverfahren sofern die Ansprüche an die Algorithmen nicht den Stand der Technik überschreiten. Neuere Ansätze wie die Full-Waveform Inversion (FWI) sind nicht Teil dessen, können jedoch über die für akademische Nutzung frei verfügbare seismische Werkzeugsammlung *SLIM* der Fakultät *Seismic Laboratory for Imaging and Modelling* der Universität British-Columbia bezogen werden [Leeuwen, 2012]. Eine Vielzahl an kommerziellen Anbietern existiert, welche sich auf den Vertrieb von hochoptimierten seismischen Algorithmensammlungen spezialisiert haben und die hier nicht weiter genannt werden sollen.

Zusätzlich zu den oben genannten Werkzeugsammlungen soll der Evaluations-Benchmark *Parboil* genannt werden [Stratton et al., 2012]. *Parboil* offeriert eine Sammlung aus seismischen Anwendungen, worunter sich einerseits die klassischen (Jacobi) *Stencil*-Algorithmen, andererseits jedoch auch Algorithmen wie die Breitensuche (BFS), Lattice-Boltzmann-Methoden (LBM) oder die einfache Matrix-Multiplikation wiederfindet. Obgleich ein derartiger synthetischer Benchmark nicht zwingend auf Demodaten angewiesen ist, so ist dieser aussagekräftiger, sofern entsprechende Daten verwendet werden. Eines der meist genutzten synthetischen Modelle ist das *Marmousi Model*, welches bereits im Jahr 1994 speziell für den Zweck der Evaluation von seismischen Algorithmen d.h. für die Tiefenmigration von Versteeg et al. entwickelt wurde

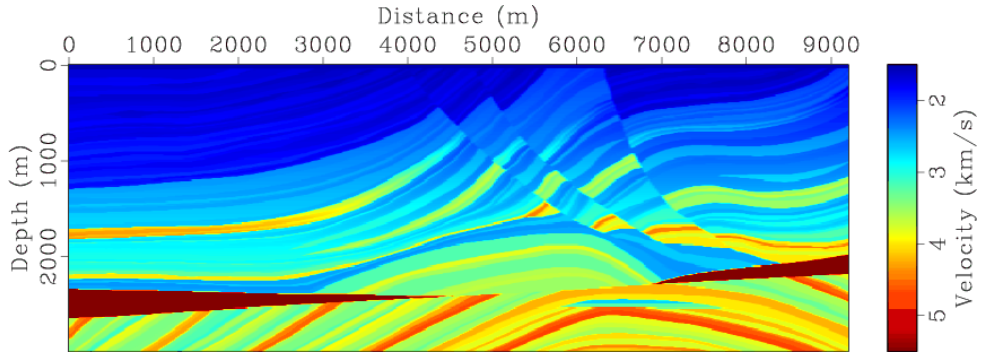


Abb. 2.10: Das synthetische seismische *Marmousi-Modell* aus dem Jahr 1994 [Versteeg, 1994].

(Abbildung 2.10) [Versteeg, 1994]. Aufgrund der weltweiten, massiven Popularität des Modells, erweiterten Martin et al. im Jahr 2006 das *Marmousi-Modell* und publizierten es als *Marmousi2* [Martin et al., 2006]. Zwar noch rein synthetisch, jedoch deutlich stärker auf Realismus getrimmt, wurden innerhalb der Jahre 2000 bis 2002 die Modelle *Pluto*, *Sigsbee* und *Ziggy* von der TU Delft erstellt, letzteres als dreidimensionales Modell des Golfs von Mexiko [TU Delft, TNO and DELPHI, 2002]. Ein weiterer Kandidat ist *SPECFEM3D GLOBE*, welcher zugleich ein vollständiges, dreidimensionales Modell der Erdkugel inklusive hochoptimierter Algorithmen bietet [Carrington et al., 2008]. Prinzipiell existieren weitere Datensätze, welche für seismische Berechnungsverfahren genutzt werden können, im Detail jedoch wurden die bekanntesten vorgestellt.

Kapitel 3

Speicherarchitekturen

Speicherarchitekturen stellen das Rückgrat eines jeden Rechensystems dar, da diese unspezifische Daten für eine bestimmte Zeit vorhalten. Je nach verwendetem Material und dessen physikalischen Eigenschaften bieten Speicher Flüchtigkeit oder Persistenz, hohe Schreib- und Lesegeschwindigkeit oder große Kapazität, geringen Energieverbrauch mit entsprechender Zuverlässigkeit jeweils in Kombination mit zu entrichtenden Kosten. Aufgrund dieses Mixes wurde eine Speicherhierarchie etabliert, welche sich den Lokalitätseffekt zunutze macht, um physikalisch nahe beieinander liegende Daten einer Rechnerarchitektur schnell bereitstellen zu können.

Trotz unterschiedlicher Speicher und insbesondere durch deren Hauptausrichtung auf Kostenminimierung, Kapazität und Packdichte, können Speicher nicht mit den (immer noch) schnellen Entwicklungszyklen von Prozessorarchitekturen mithalten. Diese Divergenz wird im Allgemeinen als die *Memory Wall* verstanden. Auch neueste Technologien wie RDRAM, HMC oder 3D Xpoint, welche gängige Formate wie DDR-DRAM- oder NAND-Flash-Speicher zu beerben vermögen, können trotz signifikanter Vorteile diese Lücke nur lindern, nicht jedoch alleinig durch eine Ersetzung überwinden. Obgleich die *Memory Wall* mit flüchtigem Speicher assoziiert wird, zeigt sich dieser Effekt auch im Bereich der nichtflüchtigen Speicher. Ein Beispiel hierfür soll das Anwendungsfeld der Biowissenschaft darstellen, bei welchem bereits digitale Mikroskope einen massiven Bedarf von mehreren Terabyte Speicherkapazität pro Tag und somit insbesondere Schreibleistung einfordern[1].

Aus diesem Grund sind neuartige Speicherarchitekturen stark im Forschungsfokus, da diese für heutige Ansprüche an ihre Grenzen stoßen [Hansson et al., 2014]. Der Forscher Jacob phrasierte hierzu im Jahr 2016, dass die Exascale-Leistung von Rechensystemen bei den Speichersystemen beginnt[2], womit er eindeutig die Beschränkung für weitere Leistungsskalierung bei den Speichertechnologien sieht [Jacob, 2016]. Der Intel-Forscher Borkar beschrieb es ähnlich brisant, indem er aussagte, dass 90 % des Entwurfs eines Höchstleistungsrechners das Speichersubsystem ausmacht und die restlichen 10 % ebenfalls dazugehören[3] [Jacob, 2016]. Deutlich früher stellte auch Cray dar, dass Speicherbandbreite nicht künstlich hergestellt werden kann, wenn sie nicht bereits vorhanden ist[4] [Sodani, 2011]. Entsprechend zeigt sich, dass Speicherarchitekturen

[1] Moderne Elektronenkryomikroskope generieren bis zu 13 Terabyte an Daten pro Tag [Rothbart, 2017]

[2] *Exascale Begins at the Memory System.* [Jacob, 2016]

[3] 90 % *of supercomputer design is the memory system ... and so is the other* 10 %. [Jacob, 2016]

[4] *You can't fake memory bandwidth that isn't there. (In the context of why the Cray-1 had no caches.)* [Sodani, 2011]

äußerst wichtig für die zukünftige Entwicklung von Rechensystemen sind.

Dieses Kapitel widmet sich daher den Speicherarchitekturen und geht ausgiebig auf den Stand der Technik von flüchtigen als auch nichtflüchtigen Speichern ein. Zugleich findet eine Vertiefung in neuartige, dreidimensionale Speicherarchitekturen statt, welche in der Forschung und der Industrie als zukünftige Standard-Speicher gehandelt werden. Auch nimmt sich dieses Kapitel der externen Verdrahtung von Speichern an; zu beobachten ist hier eine Entwicklung in Richtung einer dezentral geschalteten Bündelung. Obgleich nicht unter der Begrifflichkeit "Speicherarchitekturen" geführt, müssen Prozessorarchitekturen die Auswirkungen von Speichern mit Hilfe einer Speicherhierarchie lindern. Hierzu existieren gängige und neuartige Konzepte, welche entsprechend beleuchtet werden. Basierend auf vergangenen Trends, welche sich häufig erst in der Grafik-Domäne zeigten, widmet sich ein letztes Unterkapitel dieser Thematik genauer.

3.1 Stand der Technik

Basierend auf verschiedensten Materialien und deren Eigenschaften, Technologien und Verfahren wurde eine Vielzahl an Speichern entwickelt, welche unterschiedlich ausgeprägte Fähigkeiten in den Bereichen Flüchtigkeit, Speicherkapazität, Lese- und Schreibgeschwindigkeit (Bandbreite), Zugriffszeit (Latenz) und Energiekonsum besitzen. Heutige Speicherarchitekturen sind daher je nach Eigenschaft unterschiedlichst ausgeprägt, sodass keine als das Optimum für jedes Einsatzszenario genutzt werden kann. Aufgrund dessen existiert eine Speicherhierarchie, welche die

Hierarchie-ebene	**1**	**2**	**3**	**4**
Typ	Register	Zwischensp. (Cache)	Hauptspeicher	Hintergrundsp.
Typ. Größe	$< 1\,\text{KB}$	$< 16\,\text{MB}$	$< 16\,\text{GB}$	$> 1\,\text{TB}$
Technologie	Multiport-CMOS-Speicher	CMOS-SRAM	CMOS-DRAM	Magnetisch, CMOS-NAND
Zugriffszeit	$< 0{,}5\,\text{ns}$	$< 10\,\text{ns}$	$< 100\,\text{ns}$	$> 20\,\text{ms}$
Bandbreite	$> 20\,\text{GB/s}$	5-10 GB/s	1-5 GB/s	$\approx 100\,\text{MB/s}$
Nutzungs-steuerung	Compiler	Hardware	OS	OS/Anwender
Zugriffs-granularität	Wort	Zeile	Seite	Seite

Tab. 3.1: Illustration einer Speicherhierarchie und dessen Einfluss auf Zugriffszeit (Latenz), Bandbreite und Speicherkapazität [Buchty, 2011].

Vorteile der jeweiligen Speicherarchitektur und -technologie bestmöglich verwendet, um so Nachteile des Einzelnen zu kaschieren (Tabelle 3.1). Die Anordnung gleicht einer Pyramide, wobei kleine, flüchtige, energiehungrige, als auch teure jedoch äußerst schnelle Speicherarchitekturen den Gipfel darstellen. Langsame, nichtflüchtige, kostengünstige sowie mit hoher Speicherkapazität ausgestattete Speicherarchitekturen hingegen bilden das Fundament. Daten also Worte, die auf einem großen, persistenten, jedoch fernen und langsamen Speicher gehalten werden, werden über die Hierarchie bei Bedarf in schnellere Ebenen zwischengespeichert, wodurch in der Folge

mit jeder weiteren Hierarchieebene ein weiterer Auszug aus einem Datensatz getätigt wird. Die eigentliche Idee hinter diesem Mechanismus ist die Ausnutzung von spatialer und temporaler Lokalität; dies basiert auf der Beobachtung, dass Daten die nahe beieinander gespeichert liegen i.a. zeitnah abfolgend genutzt werden, als Daten die räumlich getrennt liegen. Innerhalb eines Rechensystems wird dies zumeist mit Hardware beschleunigt, insbesondere bei flüchtigen Speichern, wohingegen bei nichtflüchtigen Speichern zumeist auf Software zurückgegriffen werden muss.

Obgleich die geschilderte Speicherhierarchie an beliebigen Ebenen weiter vertieft werden kann, stehen gerade bei den flüchtigen Speichern kaum neue Kandidaten zur Verfügung, welche sich neben SRAM und DRAM auf dem Massenmarkt etablieren konnten. Somit wurde bereits in der Vergangenheit die Ebene der schnellen Caches vertieft, indem die Ebenen eins (L1-Cache) bis vier (L4-Cache) eingeführt worden sind, welche jeweils mit unterschiedlicher Assoziativität und Blockgrößen betrieben werden. Auch kommt manchmal auf dem gleichen Silizium eingebettetes DRAM (eDRAM) zum Einsatz, welches als L4-Cache genutzt werden kann. Beispiele hierfür sind der *Centaur* Zwischenspeicherchip der Firma IBM [Starke et al., 2015] sowie die Prozessorgenerationen Haswell und Broadwell der Firma Intel [Hammarlund et al., 2014]). Jedoch sind solch Vertiefungen im Hinblick auf die stetig wachsende Leistungslücke zwischen Speichern und Prozessor unzureichend, da diese das eigentliche Problem nicht lösen; somit werden zwingend neue Konzepte für verbesserte Speicherarchitekturen benötigt.

Obwohl die Speicherhierarchie der Stand der Technik für aktuelle Rechnersysteme darstellt, zweifeln Wissenschaftler wie Agerwala am Nutzen derer bedingt durch den Aufwand des Datentransfers und dem daraus resultierenden Energieverbrauch [Agerwala, 2014]. Zugleich offerieren neuartige Techniken wie das dreidimensionale Stapeln von Speicherbänken und der Option zum Einbetten von Rechenkapazitäten neue Wege zur Überwindung der *Memory Wall* auf.

3.1.1 Planare flüchtige Speicher

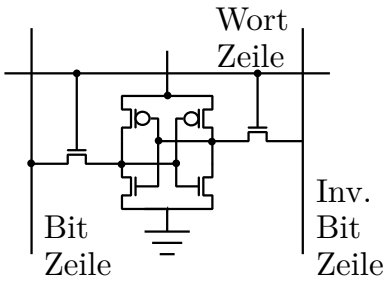


Abb. 3.1: Schema einer SRAM-Speicherzelle (engl.: *static random-access memory*, SRAM).

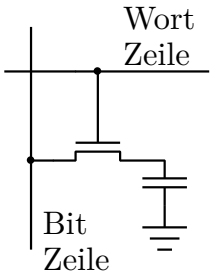


Abb. 3.2: Schema einer DRAM-Speicherzelle (engl.: *dynamic random-access memory*, DRAM).

Im Jahr 1966 erfand Dennard unter dem Titel *Field-effect Transistor Memory* eine flüchtige auf CMOS-Technologie basierende Speicherzelle, die ausschließlich aus einem Kondensator und einem Schalttransistor besteht (Abbildung 3.2) [Dennard, 1968]. Zur Speicherung eines Bits dient dabei der Kondensator, wohingegen der Schalttransistor das Lesen und Schreiben ermöglicht. Vorhergehende CMOS-Speicher nutzten mehrere Schalttransistoren, wie es auch bei der noch heutzutage verwendeten SRAM-Speicherzelle (engl.: *static random-access memory*, SRAM)

zutreffend ist (Abbildung 3.1).

Im Detail baut SRAM auf sechs Schalttransistoren auf, wobei vier davon zu zwei kreuzgekoppelten Invertern verbunden sind. Die zwei verbliebenen Transistoren werden mit Hilfe einer Wort-Leitung zum Beschreiben sowie zum Lesen benutzt. Das Schreiben findet auf die Bit-Leitung und das Lesen von der selbigen Bit-Zeile statt. Dieser Prozess geschieht bedingt durch die angelegte Spannung aktiv, während bei DRAM (engl.: *dynamic random-access memory*, DRAM) hingegen ein Verstärker (engl.: *sense amplifier*) benötigt wird, welcher die ausgelesene Ladung (d.h. *high* oder *low*) verstärkt.

Aufgrund eines anhaltenden Ladungsverlustes des Kondensators innerhalb eines DRAM-Speichers muss dieser zu regelmäßigen Intervallen neu eingelesen und geschrieben werden. Auch ein einmaliges Auslesen eines Bits kommt einem Ladungsverlust gleich, wodurch der Kondensator daraufhin wieder geladen werden muss. Im Unterschied hierzu bietet SRAM zwar schnelles Speichern und Laden ohne die Notwendigkeit der Ladungserneuerung, benötigt jedoch bedingt durch die sechs Schalttransistoren gegenüber einer DRAM-Zelle mehr Fläche und produziert entsprechend mehr Kosten.

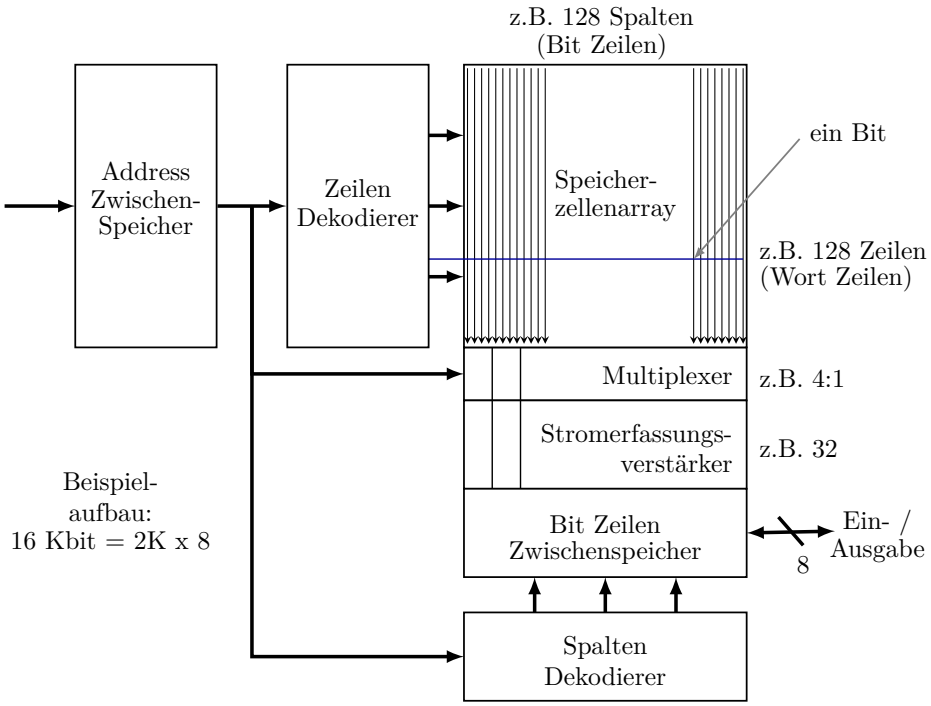


Abb. 3.3: Struktureller Aufbau einer DRAM Bank.

Eine klassische Speicherzelle benötigt weitere Komponenten, namentlich Adressdekodierer und Multiplexer, sodass eine vollständige Speicherbank, also ein Speicherzellenfeld geformt werden kann (Abbildung 3.3). Bedingt durch die spalten- und zeilenweise Anordnung der Speicherzellen entsteht zumindest theoretisch eine hochgradige Parallelität, da mehrere Bits parallel aus den Zellen ausgelesen werden können. Dagegen spricht jedoch, dass die aktuelle CMOS-Prozesstechnologie eine Verwebung von Nachbarspeicherzellen für solch eine Bank vorsieht, sodass zwar die Packdichte erhöht wird, jedoch bei sequentieller Auslesung von Zeilen direkte Nachbarn mehrere Takte nicht verfügbar sind. Hinzu kommen kleine Busse, welche eine parallel gelesene ‘Bit-Zeile’ nur in kleinen Blöcken aus der Speicherbank herausführen können.

Ein weiterer Mechanismus ist die Optimierung von DRAM auf die Anwendungsbedürfnisse hin. Beispiel hierfür ist *LPDRAM* (engl.: *low-power* DRAM) welches im Gegensatz zu regulärem DRAM eine geringere Leistungsaufnahme mit Hilfe einer geringeren Spannung, höhere Aktualisierungsrate mit partiellen Speicherzellenarray-Aktualisierungen und tiefe Stromsparmodi aufweist. Über die Jahre wurde DRAM mit verbesserter Prozesstechnologie gefertigt, was zur Folge hatte, dass höhere Taktraten im Bereich des I/O sowie der DRAM-Speicherbänke erzielt werden konnten. Zusätzlich wurde die interne Architektur angepasst, wodurch Techniken wie das *Prefetching* von (mehreren) Datenblöcken, Daten-*Burst* zur Vermeidung von Latenzen und Speicherbandbreitenmaximierung, sowie eine Übertragung von Daten bei auf- und absteigender Flanke (engl.: *double data-rate*, DDR) Einzug in DRAM hielten (je nach Generation). Weitere Techniken zur Erhöhung der Speicherbandbreite sind die Verwendung von mehreren Kanälen (siehe *dual/quad channel*), wobei im Bereich von großen Rechensystemen häufig eine Verschränkung (engl.: *interleaving*) – auch *Multi-Banking* genannt – oder eine Verwürfelung (engl.: *scrambling*) von Speicheradressen stattfindet. Bedingt durch diese Technikvielfalt hat DRAM verschiedene Generationen hervorgebracht, wovon *FPM*, *EDO*, *SDRAM*, *DDR-SDRAM*, als auch *RDRAM* und *XDR RAM* die bekanntesten Derivate sind.

Bereits in der Vergangenheit wurden weitere flüchtige Speicherarchitekturen entwickelt worden. Hierunter fallen die *Williams-Kilburn-Röhre*, das *Mellon Device* (ein optischer Speicher) oder auf *Verzögerungsleitungen* aufbauender Speicher; diese wurden jedoch durch DRAM obsolet. In der jüngeren Geschichte gab es neue Ankündigungen zu planaren, flüchtigen Speichern welche unter dem Begriff *T-RAM* sowie *Z-RAM* geführt werden.

Trotz zunehmender Konkurrenz seitens nichtflüchtiger *SCM*-Speichern (Unterabschnitt 3.1.3), stellen aufgrund ihrer Eigenschaften SRAM sowie DRAM den De-facto-Standard für Zwischenspeicher sowie Hauptspeicher dar. Im Detail findet SRAM als sehr schneller Cache für Register, Zwischenpuffer (engl.: *cache*) oder "Notizblockspeicher" (engl.: *scratchpad memory*) Anwendung (Abbildung 3.1), wohingegen DRAM als Hauptspeicher Anwendung findet (Abbildung 3.2).

Zur Beschleunigung von SRAM wurden Techniken wie z.B. dedizierte Schreib-, "Opfer"-(engl.: *victim*) sowie Strömungszwischenspeicher vorgestellt. Auch das Konzept von Bänken im Bereich von SRAM wurde evaluiert, konnte sich jedoch nicht etablieren.

3.1.2 Gestapelte flüchtige Speicher

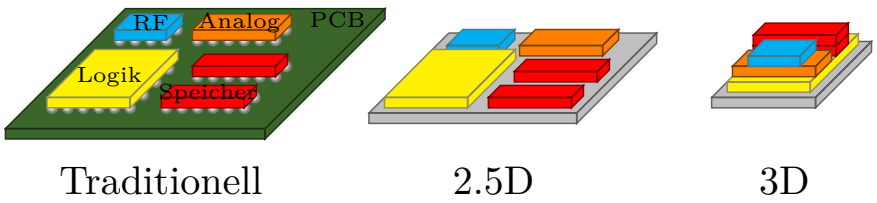


Abb. 3.4: Illustrierung einer traditionellen Verdrahtung auf einer Platine im Gegensatz zu 2.5-dimensionaler Integration und dreidimensionalem Stapeln je auf einer Siliziumverdrahtungslage.

Mit den Packtechniken der 2.5-dimensionalen Integration (engl.: *2.5D integration*) [Egawa et al., 2013] als auch der dreidimensionalen Stapelung (engl.: *3D stacking*) [Chang et al., 2013] stehen zwei vielversprechende Schlüsseltechnologien zur Verfügung, welche eine Steigerung der Lese-

und Schreibleistung d.h. Bandbreite, eine höhere Energieeffizienz, eine verbesserte Packdichte und geringere Latenzen von Speicherarchitekturen ermöglichen (Abbildung 3.4) [Patterson et al., 1997a]. Grund hierfür ist die höhere Integrations- und Packdichte, wodurch breitere Busse und höhere Taktfrequenzen derer realisiert werden können. Zugleich können unterschiedliche Fertigungstechnologien sowie unterschiedliche Materialien kombiniert werden, was zuvor entweder gar nicht oder nur unter höchsten Schwierigkeiten möglich war. Dies ermöglicht nicht nur, verschiedene Speicherarchitekturen miteinander zu integrieren, sondern auch heterogene Komponenten wie z.B. Recheneinheiten in den Speicher einzubetten. Werden daraufhin Rechenaufgaben an solch eingebettete Recheneinheiten delegiert, können externe Zugriffe eines zentralen Prozessors vermieden werden, wodurch eine höhere Effizienz ermöglicht wird. Denkbar wäre zugleich eine Abschaltung oder ein Schlafenlegen des externen Speicherbusses, nachdem eine gewisse Rechenaufgabe delegiert wurde.

Wohingegen 2.5-dimensionale Integration bereits Anwendung findet, welches auf dessen geringere Herstellungskomplexität somit geringerem Herstellungsrisiko, höherer Ausbeute, geringerem Testaufwand und verminderten Herstellungskosten zurückzuführen ist, stellt insbesondere die Thermik ein natürliches Problem der dreidimensionalen Stapelung dar [Kim und Song, 2014, Khurshid und Lipasti, 2013, Bolsens, 2011]. Obgleich bis dato nur als einzelne Techniken betrachtet, können beide auch zusammen genutzt werden, sodass ein dreidimensionaler Speicherstapel sich auf einer Siliziumverdrahtungslage wiederfinden kann. Letztere muss dabei die benötigte Durchkontaktierung (engl.: *through-silicon via*, TSV) zwischen dem 2.5-dimensionalen Substrat und dessen verschiedenen Schichten ermöglichen. Grundsätzlich setzt die 2.5-dimensionale Integration auf eine Siliziumverdrahungslage – einen sogenannten Interposer –, welche die Firma Intel jedoch mit der EMIB-Technik (engl.: *embedded multi-die interconnect bridge*, EMIB) vermeiden kann [Deo, 2015, Deo et al., 2016]. Dadurch können standardisierte Verbindungen genutzt werden. Trotz einfacherer Herstellung der 2.5-dimensionalen Integration kann die Packtechnik der dreidimensionale Stapelung als die eigentliche Schlüsseltechnologie und somit als revolutionärer Schritt angesehen werden, welche eine systemweite Energiereduktion in Kombination mit größerer Packdichte entsprechend geringerer Bauform ermöglicht [Shalf und Leland, 2015, ITRS committee, 2014].

Dreidimensionale Stapelung selbst ist ein kollektiver Begriff für verschiedenste vertikale Stapeltechniken, die in der Vergangenheit meist durch eine stufenweise “Staffelung” von Chips erzielt wurde. Diese wurden über externe, äußerst dünne Bonding-Drähte vernetzt. Hinzu kommen eine Vielzahl an weiteren vertikalen Verbindungstechniken, welche studiert oder vorgeschlagen wurden. Darunter fallen die Techniken *package-on-package* (PoP), *package-in-package* (PiP), *3D-packaged micro-bumped*, *face-to-face micro-bumped*, kontaktlos mit Hilfe von Kapazitäten oder Induktion sowie *die-to-die* [Miura et al., 2013, Davis et al., 2005, Sakuma et al., 2008]. Die einfachste Struktur weist dabei die direktkontaktierende *die-to-die* Technologie auf, da diese reguläre TSVs mit entsprechenden *Micro-Bumps* verwendet. Somit stellt diese die bevorzugte Stapeltechnik für die dreidimensionale Stapelung dar [Motoyoshi, 2009].

Marktreife, dreidimensional gestapelte Speicherarchitekturen

Die Speicherhersteller Tezzaron Semiconductors und Micron erzielten als erste das dreidimensionale Stapeln von planaren Speichern unter Verwendung der CMOS-Durchkontaktierung; die hierzu nötigen Schritte geschahen 2005 [Semiconductor, 2005] und, als wesentlicher Durchbruch,

Speichertyp	**Wide I/O**	**HMC**	**HBM**
JEDEC standard	ja	nein	ja
DRAM-Schnittstelle	Breite parallele Schnittstelle. Signalisierung ähnlich zu SDRAM. Wide I/O nutzt SDR, WIDE I/O 2 hingegen DDR.	Chip-to-chip-SERDES Schnittstelle.	Breite parallele Multikanal-Schnittstelle. DDR-Signalisierung.
Schnittstellenbreite (bits)	512 (Wide I/O), 256, 512 (Wide I/O 2).	Bis zu 4 Verbindungen mit bis zu 16 Wegen (v1.0 - v2.1). Bis zu 8 Links (v1.0, v1.1).	128 pro Kanal, bis zu 8 unabhängigen Kanälen (1024 max).
Maximale Bandbreite (GB/s)	Bis zu 17 (Wide I/O), Bis zu 68 (Wide I/O 2).	Bis zu 320 (v1.0, v1.1), Bis zu 480 (v2.0, v2.1).	Bis zu 128 (HBM), Bis zu 256 (HBM2).
Systemkonfiguration	Über Prozessor. TSV-Kontaktierung.	Punkt-zu-Punkt, geringe Reichweite mit SERDES. PCB-basiert.	2.5D TSV-basierte Siliziumverdrahtungslage (*SIP*).

Tab. 3.3: Vergleich der Speicherarchitekturstandards *Wide I/O*, *Hybrid Memory Cube* (HMC) und *High Bandwidth Memory* (HBM) [Hruska, 2015].

2011 [Pawlowski, 2011]. Am Moment sind drei in Konkurrenz stehende Produkte auf dem Markt verfügbar (Tabelle 3.3): hierbei handelt es sich um das *High Bandwidth Memory* (HBM), welches durch das Konsortium aus Hynix, AMD, Nvidia und Samsung vorangetrieben wird [Lee et al., 2015]. Direkte Konkurrenten sind der von Micron und Intel forcierte *Hybrid Memory Cube* (HMC) [Pawlowski, 2011, Jeddeloh und Keeth, 2012] sowie Samsungs Wide I/O [Hajkazemi et al., 2015]. HBM wurde als JEDEC-Standard aufgenommen und offeriert einen weiten, parallelen Bus mit einer maximalen Bandbreite von bis zu 256 GB/s (HBM2).

HMC hingegen konstituiert eine komplett neue Entwicklung, welche im spezifischen im Abschnitt 3.1.2 behandelt wird. Zusammenfassend bietet HMC eine Separierung von schneller Logik und dicht gepackten Speicherbänken, wodurch schnelle Logik eingebettet werden kann [Zhang et al., 2013]. Da die Logikschicht eines HMCs flexibel und in der Theorie nicht an Microns Standardlogik gebunden ist, könnte prinzipiell jeder Hersteller diese Logikschicht entsprechend abändern. Hieraus entstand bereits Intels spezifische Speicherarchitektur, welche unter dem Titel *Multi-Channel DRAM* (MCDRAM) geführt wird[5]. Zugleich zeigte IBM die Studie *Active Memory Cube* (AMC), bei welcher die Logikschicht gar mit Recheneinheiten ausgestattet wird [Nair et al., 2015]. Im Weiteren bietet HMC eine abstrakte Busschnittstelle sowie einen hohen internen Parallelismus bedingt durch unabhängig betreibbare interne Speicherstapel (engl.: *vault*). Wohingegen die externen Schnittstellen maximal 480 GB/s an Bandbreite ermöglichen, bietet HMC intern mit Spezifikation 2.1 eine maximale Speicherbandbreite von 320 GB/s [Hybrid Memory Cube Consortium, 2014], sodass bereits für höhere Speicherbandbreiten vorgesorgt ist.

Im Kontrast dazu wurde Wide I/O explizit für extrem energieeffiziente Speicherbandbreite zwischen einem System-On-Chip (SoC) und darauf montierten Speichern entworfen, wie z.B. für den Bereich der Smartphones. Da das dreidimensionale Stapeln Einflüsse auf Ausbeute und Validieren hat, woraus wiederum höhere Kosten resultieren, sind die aktuellen Einsatzszenarios meist im Bereich des Hochleistungsrechnens, der Grafik-Domäne, der Netzwerkgeräte sowie SoCs [Micron, 2016].

Spezifikation des *Hybrid Memory Cube*

Primär zur Stapelung multipler Speicherbankebenen sowie der Einbettung einer Logikschicht, jedoch zugleich fähig verschiedenste Technologien und Materialien dicht zu vernetzen, stellt der HMC eine exzellente Ausgangsposition dar, um Rechenkapazitäten näher an oder gar in den Speicher zu bringen. Obgleich sich bereits eine Version 3.0 in Entwicklung befindet, soll sich entsprechend dieser Abschnitt mit der aktuellen Spezifikation (Version 2.1) befassen [Hybrid Memory Cube Consortium, 2014]. Alle durch das Hybrid-Memory-Cube-Konsortium definierten HMC-Spezifikationen, d.h. Versionen 1.0, 1.2, 2.1 und 2.1, stellen einen groben Rahmen dar, anhand dessen ein (Speicher-) Hersteller einen HMC-Speicher entwickeln kann. Insbesondere die internen Details sind in Teilen frei definierbar. Zugrunde liegt ein für Speicherarchitekturen neuartiges, abstraktes Kommunikationsprotokoll, welches sich an klassischen Netzwerkprotokollen orientiert. Obgleich das Routing einfacher gebaut ist, macht dieses die traditionell speicherbuszentrischen Transferprotokolle obsolet. Als Konsequenz beschreibt jegliche Spezifikation die Verbindungsschnittstellen sowie das Netzwerkprotokoll im Detail, lässt jedoch viele Interna offen. Somit ermöglicht der HMC einen neuen Wettbewerb zwischen Speicherherstellern, da diese

[5] Aktuell wird MCDRAM ausschließlich bei dem Rechenbeschleuniger des Typs Intel Xeon Phi 'Knights Landing' eingesetzt.

	Konfiguration
Anzahl an externen Bus-Schnittstellen	2, 4
Schnittstellen-geschwindigkeit (Gb/s)	12.5, 15, 25, 28, 30
Externe Busbreite (# an *Lanes*)	4 Bit (*quarter*) 8 Bit (*half*) 16 Bit (*full*)
Speicherdichte	1 GB pro Speicherebene 4 GB (*4 Speicherebenen*) 8 GB (*8 Speicherebenen*)
Speicher je Bank	16 MB
# an Quadranten	4
# an *Vaults*	32 (*8 pro Quadrant*)
# an Partitionen	4 GB: 128 (*4 je Vault*) 8 GB: 256 (*8 je Vault*)
# an Speicherbänken	2 Bänke pro Partition
Maximal erzielbare Duplex-Busbandbreite	480 GB/s (3,84 Tb/s)
Maximale DRAM-Speicherbandbreite	320 GB/s (2,56 Tb/s)
Maximale *Vault*-Speicherbandbreite	10 GB/s (80 Gb/s)
Maximale Anzahl an vernetzbaren Speicherwürfeln	8
Maximale Anzahl an Endpunkten (SLID)	8

Tab. 3.4: Datenblatt einer HMC-Konfiguration anhand Spezifikation 2.1.

nun befähigt werden HMC standardgetreue, jedoch mit eigenen Entwicklungen versehene, z.B. optimiert auf Speicherbandbreite, Speicherlatenz oder Zuverlässigkeit, HMCs zu vermarkten. Beispiele für die Offenheit sind die einzelnen Speicherbänke, welche keine Vorgaben zu Latenzen vorgeben, oder gar die dedizierte Logikebene, welche ein Netzwerk zur Verbindung der internen Speicherstapel besitzen muss, welches jedoch nicht spezifiziert ist. Andererseits sind Frequenzen sowie Speicherbandbreiten für externe Verbindungen, sowie ebenfalls neue im Speicher auszuführende, atomare Instruktionen definiert (Tabelle 3.4).

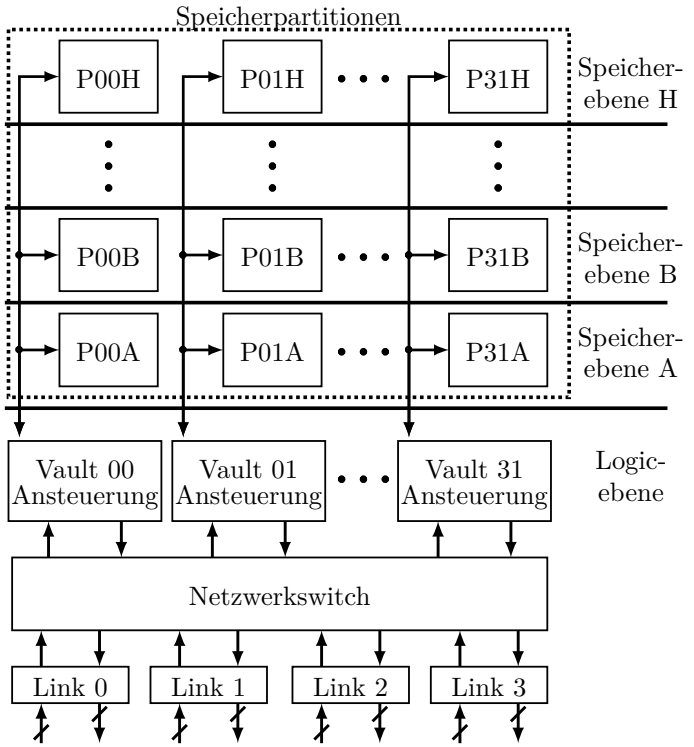


Abb. 3.5: Vereinfachte Blockdarstellung eines HMC mit vier externen Schnittstellen [Hybrid Memory Cube Consortium, 2014].

Zur Rekapitulieren der im HMC-Bereich verwendeten Akronyme dient Abbildung 3.5: Ein HMC-Speicherstapel besteht aus einer dedizierten Logikebene und mehreren, darauf gestapelten Speicherebenen. Diese Stapelung wird durch die Durchkontaktierung mittels der TSV-Technologie geschaffen [Motoyoshi, 2009]. Diese verbinden verschiedene Ebenen mit Hilfe von geätzten Löchern, welche mit leitendem Material gefüllt und so durchkontaktiert sind. Während die Speicherebenen mit einer klassischen DRAM-Prozesstechnologie gefertigt werden können, wodurch hohe Speicherdichte entsprechend Kapazität und somit geringe Kosten pro Fläche erzielt werden, kann die Logikebene wiederum mit jeglicher Technologie gefertigt werden. Wird eine Prozesstechnologie gewählt, mit welcher auch Prozessoren gefertigt werden, können hohe Frequenzen sowie schnelle Verbindungsgeschwindigkeiten erzielt werden. Die Logikebene wiederum verbindet vier sogenannte Quadranten (engl.: *quadrants*, kurz: *quads*), wobei jeder Quadrant eine eigene externe Schnittstelle (engl.: *link*) besitzen kann (Abbildung 3.6). Wie die jeweiligen Quadranten untereinander verbunden sind, ist nicht definiert, jedoch bindet jeder Quadrant zugleich sogenannte *Vaults* an (Tabelle 3.4). Diese sind innerhalb des HMC unabhängig arbeitende Speicherstapel, wobei jeder *Vault* seine eigene Speicheransteuerung (engl.: *controller*) für die darüber gestapelten Speicherbänke - die sogenannten Partitionen - besitzt. Die HMC-Spezifikation

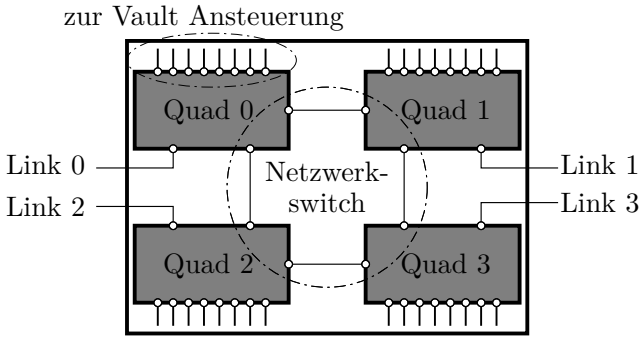


Abb. 3.6: Illustration einer theoretisch validen HMC Logikebene.

2.1 sieht insgesamt acht *Vaults* je Quadrant vor, wohingegen jeder Quadrant zusätzlich eine externe Schnittstelle besitzt.

Jede Speicherpartition besitzt insgesamt zwei Speicherbänke, wobei jede Speicherbank mit ECC-Fehlererkennung ausgestattet ist. Dies stellt insofern ein Novum dar, da ECC-Fehlererkennung meist nur in Spezialfeldern wie in Servern (ECC DIMM) eingesetzt wird. Werden HMC-Speicherstapel mit einem Host wie z.B. einen Prozessor verbunden, so werden diese Endpunkte als SLID (engl.: *source-link identifier*, SLID) bezeichnet. Obgleich reguläre Schnittstellen daraufhin als SLID tituliert werden, muss eine Schnittstelle nicht der SLID-Nummerierung folgen. Für ein Netzwerk aus acht verbundenen HMC-Speicherstapeln sind jeweils acht SLID vorgesehen. Gerade im Bereich des externen Busses wird die Busbreite meist mit *Lanes* und somit Spuren bezeichnet, wobei jede *Lane* ein Bit pro Takt übermitteln kann.

Da auch die interne Frequenz vom Speicherhersteller selbst gewählt werden kann, wird diese nicht explizit in der Spezifikation angegeben. Gemäß Literatur betreibt Micron seine Referenz HMC-Implementierung mit 1,25 GHz (t_{ck} = 0,8 ns) [Rosenfeld, 2014, Zhang et al., 2015]. Dies bedeutet, dass unter Annahme einer Taktfrequenz von 1,25 GHz, sechzehn *Lanes* (Busbreite in Bit) und einer maximal möglichen Bitrate von 30 Gbit/s jede Schnittstelle bidirektional 480 GB/s erzielt, wodurch exakt 384 Bits (48 Bytes) pro Takt übertragen werden. Zugleich wird in der Literatur angegeben, dass je ein *Vault* eine maximale Bitrate von 2,5 Gbit/s bietet, wobei von einem internen Businterface von 32 Bits (32 TSV-Daten-*Lanes*) ausgegangen werden kann. Diese würde einer Bandbreite von 64 Bits (8 Bytes) pro Takt je internem sowie unabhängigen Speicherstapel entsprechen.

3.1.3 Nichtflüchtige Speicher

Gegenwärtig kann ein großes Spektrum an etablierten als auch neuartigen, nichtflüchtigen Speichern (engl.: *non-volatile memory*, NVM) ausgemacht werden. Bei NVMs handelt es sich um persistente Speicher, welche Daten auch nach Ausschalten eines Rechners beibehalten. Bereits zu Frühzeiten setzten Rechensysteme auf mechanische nichtflüchtige Speicher, um diese als Hauptspeicher (siehe Magnettrommel) oder als Festwertspeicher (engl.: *read-only memory*, ROM) zum Initialisieren (*Boot*-ROM) zu verwenden. Im Verlauf der Zeit wurden jedoch mechanische Speicher aufgrund ihrer Lese- und Schreibgeschwindigkeit sowie Zugriffszeit durch schnellere, flüchtige Speicher ersetzt. Entsprechend kommen nichtflüchtige Speicher, welche auf magnetischen, optischen oder mangneto-optischen Eigenschaften beruhen, heutzutage als Hintergrund- oder

gar als Archivspeicher zum Einsatz. Einmalig fest zu verdrahtende und löschbare Festwertspeicher wie programmierbare ROM (engl.: *programmable ROM*, PROM), UV-löschbare PROM (engl.: *erasable PROM*, EPROM) sowie elektrisch löschbares PROM (engl.: *electrically EPROM*, EEPROM) widerfährt wiederum ein Ersatz durch NAND-Flash-Speicher.

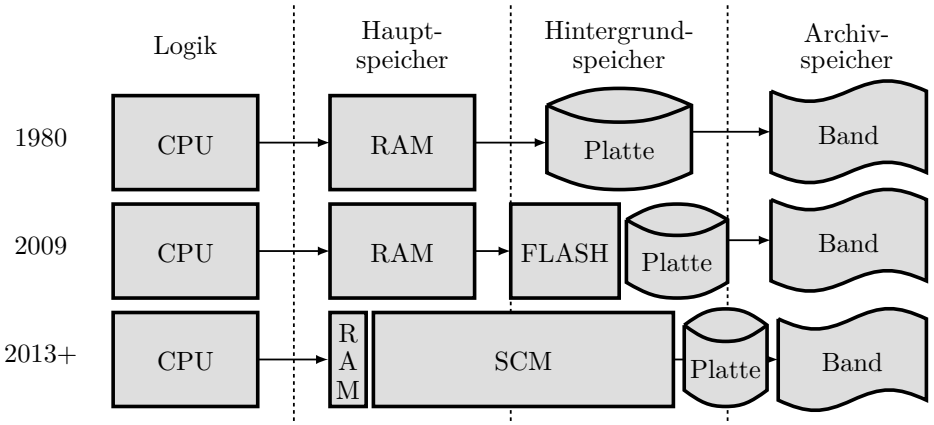


Abb. 3.7: Evolution der Speicherhierarchie bedingt durch NAND-Flash-Speicher, welcher die traditionellen Grenzen zwischen Hauptspeicher und Hintergrundspeicher direkt, sowie Hintergrundspeicher und Archivspeicher indirekt verwischt [Doller, 2016].

NAND-Flash-Speicher verdrängen jedoch nicht nur Festwertspeicher, sondern ermöglichen aufgrund hoher Ein- und Ausgabe-Befehle pro Sekunde (engl.: *input/output operations per second*, IOPS), moderater Kosten und großer Speicherkapazität (bei geringer Zuverlässigkeit) die Teilüberbrückung der *Memory Wall* im Bereich zwischen magnetischen Hintergrund- sowie flüchtigen Hautspeichern (Abbildung 3.7). Ein Beispiel hierfür stellt die Technik "vertikaler NAND" (engl.: *vertical NAND*, V-NAND) der Firma Samsung dar, welche mit Charge-Trapping-Speicher und vertikaler Stapelung eine äußerst hohe Packdichte entsprechend Kapazität, Langlebigkeit sowie Lese- und Schreibgeschwindigkeit erzielt [Samsung, 2014].

Angesichts dessen wird NAND-Flash als erster Kandidat der neuen Schlüsseltechnologieklasse der *Storage Class Memories* (SCM) gehandelt, welche das Potential besitzen einerseits als Hauptspeicher, andererseits als Hintergrundspeicher genutzt werden zu können. Trotz der Verdrängung der etablierten Konkurrenz sind auch heutzutage mechanische Speicherarchitekturen bedingt durch ihre deutlich höhere Speicherkapazität in Kombination mit signifikant günstigeren Kosten im Bereich der Festspeicher überlegen. Somit sind diese noch heute Teil der Speicherhierarchie. Aufgrund dessen wird selbst noch im Bereich von Magnetband aktiv geforscht [Hemsoth, 2017b]. Ein Beispiel hierfür stellt der von IBM und Fujifilm im Jahr 2015 aufgestellte Rekord mit dem sogenannten LTO6-Magnetband dar, bei welchem 220 Terabyte auf eine handtellergroße Magnetbandkassette gespeichert werden konnte [IBM, 2015]. Hierbei zeigt sich, dass die Speicherkapazität alle zwei Jahre verdoppelt werden kann.

3.1.4 Neuartige nichtflüchtige Speicher

Neuartige nichtflüchtige Speicherarchitekturen versuchen die Mängel hinsichtlich Speicherbandbreite und Speicherlatenz innerhalb der Speicherhierarchie zu überwinden. Obgleich derartige Speicher sich vornehmlich zwischen flüchtigen und magnetischen Speichern etablieren, ist die Ersetzung von flüchtigen Speichern wie DRAM das ausgemachte Ziel, womit sich in Folge große

Speicherkapazitäten als Hauptspeicher etablieren ließen. Ein Vorteil wäre, dass hiermit Rechensysteme innerhalb von wenigen Millisekunden aus einem ausgeschalteten Zustand wieder ihren Betrieb aufnehmen könnten.

Bedingt durch verschiedene Technologien und Materialien zeigen neuartige nichtflüchtige Speicherarchitekturen Stärken und Schwächen in den Bereichen Energieeffizienz, Zuverlässigkeit, Speicherzellenfläche sowie Kosten [Gara, 2012, ISSCC, 2017, Xie, 2013]. Ein Beispiel hierfür ist das im Jahr 2015 von Intel und Micron angekündigte Produkt *3D Xpoint*, welches nichtflüchtigen Speicher mit großer Speicherkapazität in Kombination mit hohen *IOPS* verspricht [Coughlin, 2016]. Dies soll die Lücke zwischen flüchtigem Hauptspeicher und NAND-Flash-Speicher schließen. Da *3D Xpoint* zugleich als persistenter Hauptspeicher genutzt werden kann, bietet der Speicher einerseits eine Byte-Adressierung und - wie in flüchtigen Speichern üblich - eine Seitenadressierung mit 4 KB oder 64 KB an. Technisch baut *3D Xpoint* auf *Phase-Change-Memory* auf [Choe, 2017]. Obgleich eine auf Intels *3D Xpoint* aufbauende *NVMe* SSD mit dem Titel Optane in ersten Benchmarks einen hohen Lese- und Speicherdurchsatz von bis zu 2 GB/s erzielte, sind die Speicherkapazitäten noch äußerst beschränkt und die Kosten recht hoch [Intel, 2017, Handy, 2017]. Die Konkurrenten SanDisk und HP kündigten im Rahmen von Intels *3D Xpoint* Vorstellung einen bis dato undefinierten SCM-Speicher an, welcher ähnliche Eigenschaften wie *3D Xpoint* bieten soll [SanDisk, 2015].

Während NAND-Flash-Speicher sowie *3D Xpoint* bedingt durch eine Marktverfügbarkeit aktuell die meiste Aufmerksamkeit erfahren, werden weitere zukünftige Kandidaten erforscht oder sind bereits mit äußerst kleinen Kapazitäten verfügbar, wie z.B.:

- *Conductive-bridging RAM* (*cbRAM* bzw. *programmable metallization cell* (*PMC*) Speicher)
- *Conductive metal-oxide* (*CMOx*) Speicher
- *Ferroelectric-gate RAM* (*FeRAM*/*FRAM*)
- *FJRAM* (*Floating Junction Gate Random Access Memory*)
- *Magnetoresistive random-access memory* (*MRAM*), bzw. *spin-transfer-torque magnetic RAM* (*STT-MRAM*)
- *Phase-change memory* (*PCM*/*PCMe*/*PRAM*/*PCRAM* bzw. *Chalcogenide RAM*/*CRAM*)
- Polymere Speicherarchitekturen
- *Resistive RAM* (*ReRAM*/*RRAM*) (bzw. der *Memristor*)
- *SONOS* (*Silicon-Oxide-Nitride-Oxide-Silicon*) Speicher

In Anbetracht der Vielzahl kann zum aktuellen Zeitpunkt eine kondensierte Auflistung erstellt werden, welche die aussichtsreichsten Kandidaten für die kommenden Jahre darstellt (Tabelle 3.5). Hierunter finden sich u.a. der bereits angesprochene *3D Xpoint*, der *Memristor* (*ReRAM*), *Z-NAND* sowie verschiedene Variationen an *MRAM*. *MRAM* bietet hinsichtlich heutiger flüchtiger DRAM-Speicher in Bezug auf die Speicherbandbreite das Potential DRAM zu beerben. Jedoch ist keiner der Kandidaten im Bereich der Speicherlatenz DRAM ebenbürtig,

Speicher	Hersteller	Anbindung	Formfaktor	Latenz	Bandbreite
MRAM	Everspin	DDR3 / DDR4	DIMM	10 ns/20 ns	30 GB/s
3D Xpoint	Intel, Micron	PCIe, NVMe	SSD, DIMM	50 ns/1 us, ≈200 ns	1,6 GB/s, 12,8 GB/s
3D PCM	SEC, SK-Hynix	DDR4 / DDR5	DIMM	≈200 ns	12,8 GB/s
ReRAM	Toshiba, WD	DDR4 / DDR5	DIMM	100 ns/100 us	12,8 GB/s
Z-NAND	SEC	DDR4	SSD, DIMM	12≈20 us, 5≈8 us	8 GB/s, 12,8 GB/s
(48-lagiger) 3D V-NAND	Samsung	NVMe	SSD	≈24 us	2,1 GB/s (wr) 3,5 GB/s (rd)
DDR4-2133	*Diverse*	DDR4	DIMM	≈1 ns	17 GB/s

Tab. 3.5: Die aussichtsreichsten SCM-Speicherarchitekturkandidaten (zum Teil noch in Entwicklung oder neu erschienen sind). Im Vergleich hierzu aktueller 3D V-NAND-Flash-Speicher und DDR4-DRAM [Ranjan, 2016].

sodass weitere Forschung erforderlich ist. Andererseits bieten alle Technologien explizit Persistenz, wodurch große Rechensysteme bei einem Neustart DRAM signifikant überlegen sind. Im Gegensatz zu DRAM stellt moderner, auf dem Markt bereits verfügbarer NAND-Flash-Speicher für alle neuartigen, nichtflüchtigen Speicherarchitekturen prinzipiell keine Hürde dar.

Da der *Memristor* neue Dimensionen für Speicher- und Rechenarchitekturen eröffnen könnte, soll hiermit noch explizit darauf eingegangen werden. Im Jahr 1971 zeigte L. Chua Indizien auf, dass ein viertes Schaltelement existieren könnte, namentlich der *Memristor*, welcher neben Kondensator, Spule und Widerstand existieren soll [Chua, 1971]. Ein *Memristor* wäre eine tatsächliche Revolution für Speicherarchitekturen, da er Bits permanent aufgrund von Widerstand und nicht durch eine Kapazität speichern würde.

3.2 Konzepte zur geteilten Speicherarchitekturanordnung

Durch die Vernetzung von multiplen Speichermodulen kann ein großer gemeinsamer Speicherpool geformt werden. Dieser kann innerhalb eines Rechensystems multiplen Recheneinheiten zur Verfügung gestellt werden; solch Rechensystem wird geteilte Speicherarchitektur (engl.: *shared memory architecture*) genannt. Eine dieser Ausprägungen ist die UMA-Architektur (engl.: *uniform-memory access*, UMA), bei welcher alle Recheneinheiten einen solchen gemeinsamen Speicherpool einheitlich nutzen können. Einheitlich meint hier, dass alle beteiligten Recheneinheiten den gleichen Adressraum sehen und mit gleicher Zugriffszeit auf die verteilten Speicherbänke zugreifen können (Abbildung 3.8). Beispiele für diese Konstruktion sind vielfältig vorzufinden, u.a. bei eingebetteten Systemen, Desktops, Smartphones als auch Servern.

Im Kontrast dazu steht die NUMA-Speicherarchitektur (engl.: *non-uniform memory access*, NUMA). Ein gemeinsamer Adressraum ist bei der NUMA-Speicherarchitektur ebenfalls für beteiligte Recheneinheiten gegeben, jedoch existieren mehrere Speicherpools. Solche Speicherpools

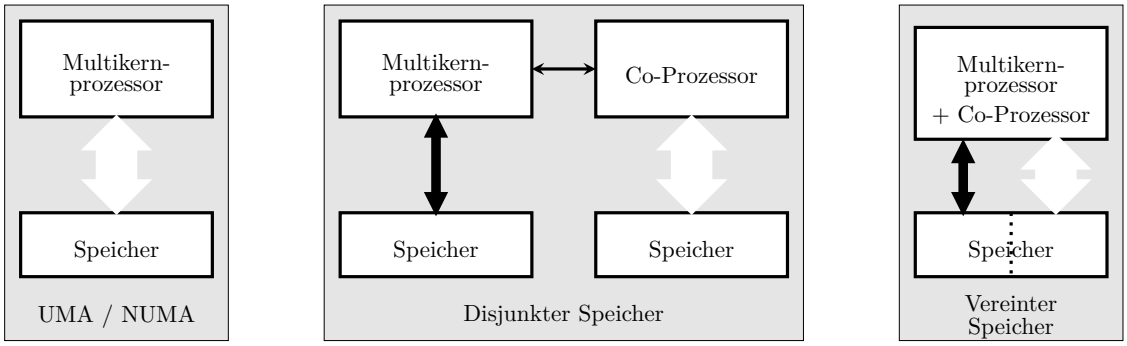


Abb. 3.8: Drei Konzepte zur Anbindung von Speicher an Recheneinheiten. Je breiter die Pfeile zwischen Speicher und Prozessor, je breiter fällt der Bus aus und somit größer ist die Speicherbandbreite.

sind zwar miteinander verbunden, jedoch ist diese Verbindung meist langsamer und nicht unbedingt kohärent. Dadurch sehen Recheneinheiten für lokalen Speicher kürzere Zugriffszeit als für fernen Speicher. Der Vorteil besteht darin, dass eine höhere Speicherzugriffsparallelität bestehen kann, insofern Recheneinheiten vermehrt auf ihren lokalen und nur wenig auf fernen Speicher zugreifen. Für NUMA-Speicherarchitekturen nachteilig wirkt sich jedoch die Speicherkohärenz aus, da diese in Hardware äußerst schwierig zu realisieren ist. Sind solch verteilten Speicherpools kohärent, werden sie unter dem Begriff ccNUMA (engl.: *cache-coherent NUMA*, ccNUMA) geführt.

Eine Vielzahl von nicht kohärenten NUMA-Speicherarchitekturen existiert wie etwa heterogene Rechensysteme, welche Prozessoren mit Beschleunigern kombinieren. Derartige Ansätze finden zunehmend auch ihren Weg in eingebettete Systeme, etwa Adapteva Parallella [Olofsson et al., 2011] oder Rex Computing NEO [Sebexen und Sohmers, 2015]. Bedingt durch die Popularität von heterogenen Systemen zeichnet sich eine Entwicklung bei NUMA-Speicherarchitekturen ab, bei welcher ein externer Bus trotz schwieriger Realisierbarkeit kohärent, d.h. *ccNUMA*-konform, gemacht wird. Beispiele hierfür sind IBM mit der Technik (Open)CAPI, Nvidia mit NVlink, Intel mit QPI und das OpenFabrics Consortium mit CCIX und GEN-Z [Benton, 2017]. Einen Schritt weiter geht die Firma HP mit dem Rechensystem *The Machine*, welches die NUMA-Problematik global betrachtet: Dies bedeutet, dass verteilte Rechensysteme einen gemeinsamen, direkt adressierbaren und kohärenten Speicher anbieten, wobei einzelne Partitionen innerhalb dessen jeweils von einem Rechenknoten bereit gestellt werden [Hewlett Packard Enterprise, 2016]. Das Konzept ist innerhalb der Softwareprogrammierung unter dem Titel *Partitioned Global Address Space* (PGAS) nicht neu, wird nun jedoch bei *The Machine* explizit als hardwaregestützte verteilte Speicherkohärenz angeboten. Somit stellt *The Machine* eine verteiltes *ccNUMA*-Rechensystem dar.

Zusätzlich existiert das Konzept CoMA (engl.: *cache only memory architecture*, CoMA), bei welchem ausgehend von einer NUMA-Speicherarchitektur jeder einzelne Hauptspeicher derer wiederum als Cache angesehen wird. Die Adressierung wird dabei über einen Streualgorithmus (engl.: *hashing*) realisiert, sodass jede sequentiell hintereinander zu adressierende Cachezeile nicht zwingend im selben Speicher liegt. Wird eine Cachezeile per Laden angefordert, so verschiebt sich deren "Heimspeicher" ebenfalls, worüber der Hash-Algorithmus entsprechend sensibilisiert werden muss.

3.3 Auswirkung auf Prozessorarchitekturen

Zugespitzt formulieren Rogers et al., dass industrielle Hersteller in der Vergangenheit ausschließlich die Frequenz von externen Bussen sowie die Anzahl von Kanälen derer vergrößert haben, die eigentlichen Speicherarchitekturen jedoch hinsichtlich der *Memory Wall* missachteten [Rogers et al., 2009]. Dies, so Rogers et al., ist für zukünftige Entwicklungen nicht mehr tragbar, da die Frequenz durch die *Power Wall* limitiert wird, während die Zunahme an Buskanälen, entsprechend Verdrahtung zugleich durch die Platinenfläche, Platinenlagen, Kosten sowie thermische Aspekte beschränkt ist. Traditionell wurde daher eine Vertiefung der Speicherhierarchie in Kombination mit einer intelligenten Verwendung der *Harvard-Architektur* vollzogen, sodass diese nicht nur auf die erste SRAM-Zwischenspeicherebene sondern auch auf weiteren Ebenen Anwendung fand (Abschnitt 3.1). Während in den Frühzeiten der Prozessorarchitekturen, Zwischenspeicher als externe Bausteine zwischen zur damaligen Zeit als Hauptspeicher verwendeten, rotierenden Trommeln fungierten, wurden diese vermehrt in Prozessoren integriert. Bedingt

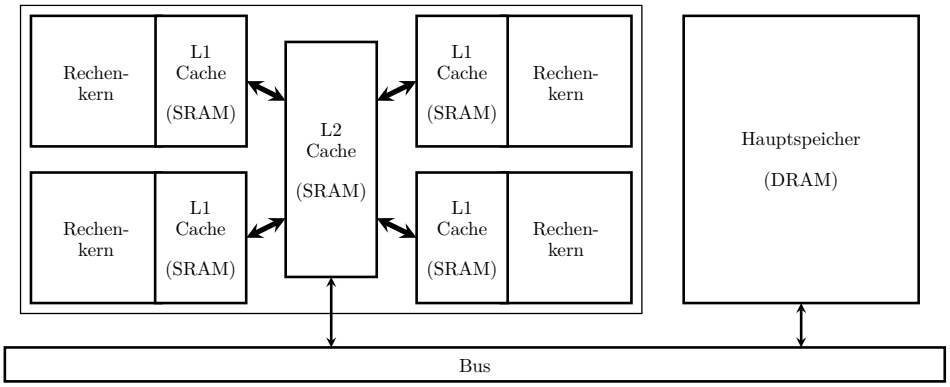


Abb. 3.9: Schema einer modernen Multikern-Rechnerarchitektur, bei welcher die Speicherhierarchie mit den dedizierten Speicherarchitekturen dominiert.

durch den aus der *Memory Wall* resultierenden *Von-Neumann-Engpass*, bildet in heutigen Multikernprozessoren meist der Bus die wesentliche zentrale Komponente, an welcher sich alle weiteren Architekturen auszurichten haben (Abbildung 3.9). Dabei folgen zuerst Speicherarchitekturen, welche die Speicherhierarchie mit verschiedenen SRAM-Ebenen abbilden, gefolgt von den Prozessorarchitekturen.

3.3.1 Milderung der Memory Wall

Um die Auswirkungen der *Memory Wall* zu minimieren, wurde eine Vielzahl an Techniken entwickelt, welche im Folgenden beschrieben werden soll. Die meisten machen sich Lokalitätseffekte zunutze oder verwenden ausgefeilte Hardwaretechniken, um hohe Latenz und geringe Speicherbandbreite zu kaschieren. Dies benötigt meist signifikant Logik, was gleichzeitig Fläche bedeutet, wodurch beim Entwurf neuer Architekturen eine Kosten- / Nutzen-Abwägung angewandt werden muss.

Eine Technik stellen viele Kerne eines Prozessors oder aber viele Hardware-Threads dar. Durch die Masse an Ressourcen kann die Speicherbandbreite einfach saturiert werden. Ist einer der Hardware-Threads nicht ausführbar, kann ein neuer Kontext eingeladen werden, welcher idealerweise rechenbereit ist. Beispiele hierfür sind der Höchstleistungsrechner IBM BlueGene/Q,

die beiden Cray Rechensysteme MTA und XMT [Kopser und Vollrath, 2011] sowie Grafikkarten, welche diesen Effekt signifikant ausnutzen. Gerade der Cray MTA wendet dieses Konzept auf einen global über verteilte Rechenknoten adressierbaren Speicherraum an, damit die zusätzliche Latenz des Netzwerks minimiert werden kann. Prinzipiell ist es auch möglich, im Bereich des Software-Threadings Registerfenster, wie sie in der SPARC-ISA vorzufinden sind, ebenfalls zur Latenzminimierung zu verwenden, da bei einer ausstehenden Speicheroperation der Kontext gewechselt werden kann [Weaver und Germond, 1994]. Aus der Verwendung von einer Vielzahl an kleinen Rechenkernen folgert Rogers et al. jedoch, dass solche Techniken trotz allgemeiner Anwendung meist nur einen moderaten Rechenleistungsgewinn bieten, da diese die *Bandwidth Wall* nur indirekt adressieren [Rogers et al., 2009]. Entsprechend kategorisieren Rogers et al. die Speicherbandbreite verbessernde Techniken in direkt, indirekt und dual, wobei jegliche Caches sowie *Scratchpads* auch ausgestattet mit unabhängig arbeitenden *Direct Memory Access* (DMA) Einheiten unter indirekte Methoden fallen [Kahle, 2005].

Ein weiteres Mittel zur Bandbreitenmaximierung stellen Register dar, welche z.B. als (SIMD-) Vektorregister vorliegen können (Unterabschnitt 2.1.4) [Nair et al., 2015]. Auch wenn es unter Umständen mehrere Takte benötigt, derartige Register zu füllen, sind zu einem gewissen Zeitpunkt mehr Daten innerhalb des Prozessors. Dies lindert zugleich die Menge an zu ladenden Instruktionen, da für eine Vektorinstruktion mehrere skalare Instruktionen überflüssig werden. Zumeist verhelfen Cachezeilen ausgerichteten Vektorlade-/speicheroperationen zusätzlich, die voll verfügbare Speicherbandbreite zu saturieren. Auch zusätzliche Register in einer Rechnerarchitektur verhelfen dazu, mehr Daten vorzuhalten. Problematisch ist hier jedoch, dass zusätzliche Bits zur Registeradressierung benötigt würden. Im starken Kontrast dazu steht Thoziyoors registerlose Prozessorarchitektur, bei welcher nicht nur für jegliche Operation, sondern auch für jedes Datum, das sonst von einem Register gespeichert wird, eine Ladeoperation entsteht [Thoziyoor et al., 2005]. Dies ist explizit gewollt, da jegliche Operation einen Kontextwechsel nach sich zieht.

Tiefe Lade- und Speicherwarteschlangen erlauben es, multiple ausstehende Ladeoperationen (engl.: *in-flight operations*) vorhalten zu können [Cristal et al., 2005]. Entsprechend kann Maschinencode mit wenigen Registerabhängigkeiten im besten Fall ohne Pipeline-Stall verarbeitet werden [Nair et al., 2015]. Hervorgehoben von Saulsbury et al. verhelfen die klassische Operationsausführung in nichtlinearer Reihenfolge (engl.: *out-of-order execution*) mit Hilfe des *Re-Order Buffers* (ROB) zu weniger Registerabhängigkeiten [Saulsbury et al., 1996] sowie eine intelligente Software-Disposition von Ladeoperationen, Latenzen zu minimieren [Kagi et al., 1996]. Im Bereich der Lade- und Speicherwarteschlangen und der *Out-Of-Order Execution* wurde im Rahmen der Kilo-Instruktions-Mikroprozessoren äußerst intensive Forschung betrieben. Ergebnisse hierzu sind der *Slice-And-Data Buffer* (SDB) [Jothi et al., 2011], der *Pseudo-Re-Order-Buffer* (Pseudo-ROB) als auch die "Kriechspur-Instruktionswarteschlange" (engl.: *slow-lane instruction queue*, SLIQ) [Cristal et al., 2004]. Im Detail werden Hardware-Operationen aus dem ROB in den SDB überführt, sofern diese abhängig von einer ausstehenden Ladeoperation sind. Dies ist immer dann der Fall, wenn diese nicht vom Cache bedient werden konnten. Somit werden Reservierungsplätze im ROB für unabhängige Programmteile freigegeben. SLIQ hingegen verschiebt langlaufende, ausstehende Ladeoperationen in eine dedizierte Warteschlange, um unabhängigen Operationen eine potentielle Chance auf Ausführung zu ermöglichen.

Obgleich die Ausnutzung von Lokalität innerhalb der Caches zu einer hohen Trefferquote (engl.: *hits*) führt, kann ein Vorabrufen (engl.: *prefetching*) entweder per Software oder gar durch

Hardware in Form von *bus-snooping* zu geringerer Latenz führen. Als Beispiel bietet der IBM BlueGene/Q einen programmierbaren L2-Prefetcher, welcher bei einem erstmaligen Verarbeiten von Zugriffen die ausstehenden Cachezeilen Anfragen aufnehmen kann. Daraufhin wird eine Wiedergabe gestartet, welche auch bei Zugriffen auf dünn besetzte Strukturen erfolgreich die nachfolgend benötigten Cachezeilen vorab lädt [Haring et al., 2012]. Gleiche Effekte bietet eine spekulative Ausführung, worunter z.B. das Konzept des transaktionsbasierten Speichers (engl.: *transactional memory*) fällt [Kagi et al., 1996, Cristal et al., 2005].

Dem Konzept von Rogers et al. folgend, schlugen Burger et al. die direkte Technik der Linkkompression zwischen Prozessor und Speicher sowie die indirekte Möglichkeit der Verknüpfung von beiden auf einer gemeinsamen Siliziumlage vor [Kagi et al., 1996]. Da zur damaligen Zeit technologisch nicht realisierbar, wurde zugleich ein dreidimensionales Stapeln von solch Komponenten vorgestellt. Stanley-Marbell et al. untermauerten diesen Aspekt und fügten *Silicon Photonics*, d.h. zum Datenaustausch die Integration von Optik in Silizium, und integrierte Spannungsregulatoren hinzu[6] [Stanley-Marbell et al., 2011]. Dadurch sollte die Problematik der Kontaktstifte zwischen Platine und Chip gelindert werden. Zusätzlich stellten Rogers et al. multiple weitere Aspekte vor, darunter eine Cache-Kompression, die Integration eines DRAM-Caches über einem SoC, das dreidimensionale Stapeln mehrerer Cacheebenen (im Gegensatz zu Burger et al.), das Filtern von nicht genutzten Daten bei Abruf einer kompletten Cachezeile sowie sektorbasiertes Zwischenspeichern [Rogers et al., 2009].

3.3.2 Orthogonale Entwicklungen in der Forschung

Das Gehirn als Vorbild für massiv asynchrone Verarbeitung inspiriert neuartige parallele Rechnerarchitekturen, welche sich das menschliche Lernen und somit das dynamische Vernetzen von Neuronen durch Synapsen zunutze macht. Solch neuromorphen Architekturen erlernen und vergessen, und bieten auf kleinstem Raum hochspezialisierte “Berechnungen” bei äußerster Energieeffizienz. International wurden multiple Projekte gefördert, worunter die durch die DARPA geförderten Projekte *TrueNorth* (*SyNAPSE*, IBM) und *Neurogrid* (Universität Standford), sowie die beiden im Rahmen der EU-Initiative *Human Brain Project* geförderten Projekte *SpiNNaker* (Universität Manchester) und *HiCANN* (Universität Heidelberg) fallen [Furber, 2016]. Die Genannten stellen die erste Generation von das Gehirn simulierenden Architekturen dar, sodass jede Plattform Einschränkungen bgzl. Anzahl an Neuronen und Synapsen, der Energieeffizienz als auch der Programmierbarkeit aufweist. Furber weist zusätzlich auf den Lernprozess hin, welcher sich mit Ausnahme von *HiCANN* im Rahmen der Geschwindigkeit des menschlichen Lernens bewegt. *HiCANN* hingegen bietet ein 10.000-fach schnelleres Lernen und damit die Möglichkeit, Jahrzehnte an Lernen innerhalb von Stunden zu bewältigen [Furber, 2016]. Mit Ausnahme der Lerngeschwindigkeit und der Programmierbarkeit übertrifft jedoch IBMs neuromorpher CMOS-Chip *TrueNorth* seine Konkurrenz, da die in Hardware modellierten asynchronen Synapsen ereignisorientiert statt klassisch mit Prozessorfrequenz arbeiten, als auch ein On-Chip-Netzwerk sowie eine fortschrittliche, aus dem mobilen Bereich stammende 28 nm-Niederspannungs-Digital-Technologie genutzt wurde [Merolla et al., 2014]. Bereits *TrueNorths* Layout weist im Kontrast zu traditionellen Architekturen signifikante Unterschiede auf, darunter eine nicht existente Cachehierarchie, Einbettung von DRAM sowie eine dedizierte Terminierungslogik und keine explizit

[6] Intels Haswell-Architektur bot voll integrierte Spannungsregulatoren an (engl.: *fully integrated voltage regulators*, FIVR) [Burton et al., 2014].

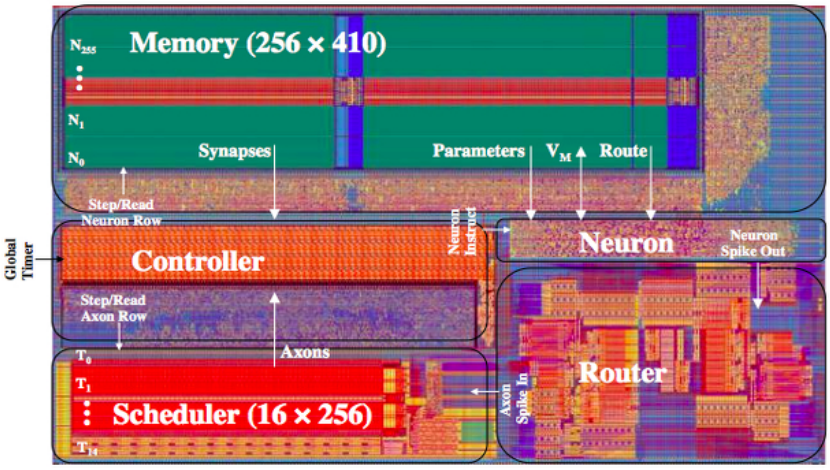


Abb. 3.10: Layout des neuromorphen CMOS-Chip *TrueNorth* der Firma IBM [Merolla et al., 2014].

vorhandene ALU (Abbildung 3.10). Zusätzlich wurde ein hybrides System aus neuromorpher *TrueNorth-* und *Von-Neumann-Architektur* geschaffen, um einerseits das Potential beider parallel bestmöglich zu nutzen und andererseits eine vereinfachte Ein- und Ausgabe zu ermöglichen.

Nicht unbedingt als neue Entwicklung, jedoch als ein neues Anwendungsfeld aus der Nische des Quantenrechnens (engl.: *quantum computing*) wurde bereits im Jahr 2015 durch die Kooperation zwischen Microsoft und Rambus ein Quantenspeicher unter dem Titel *Cryogenic Memory* angekündigt [Yoshida, 2017]. Dieser basiert primär auf einer regulären CMOS-Technologie (DRAM), obgleich diese nicht für den Betrieb bei einer Tiefsttemperatur von 77 Kelvin vorgesehen ist. Für die Zukunft ist angedacht, dass Quantenrechner direkt mit *Cryogenic Memory* betrieben werden können. Im Gegensatz dazu wurden im Jahr 2017 ionenbereicherte Kristalle vorgestellt, bei welchem ein Photon ein Qubit darstellt [Laplane et al., 2017]. Wird dieses in den Kristall überführt, wird es durch multiple Ionen absorbiert, welche das entsprechende Photon daraufhin emittieren. Dies stellt eine Speicheroperation dar. Bei einem Beschuss mit einem Laser wird das Photon wiederum aus den Ionen herausgelöst, wodurch eine Ladeoperation abgebildet werden kann.

3.4 Zusammenfassung

Aufgrund der sich zuspitzenden *Memory Wall* finden sich Speicherarchitekturen in einem äußerst angestiegenen Forschungsinteresse wieder. Betroffen hierbei sind gleichsam flüchtige und nichtflüchtige Speicherarchitekturen, wobei bereits ein Wandel weg von gängigen Formate wie flüchtigen planaren DRAM-Speicher oder magnetischen Festspeicher hin zu flüchtigen dreidimensionalen *High Bandwidth Memory*, *Hybrid Memory Cube* sowie NAND-Flash-Speicher begonnen hat. Dabei wird meist eine Vertiefung der Speicherhierarchie vollzogen, wodurch die etablierten als tiefere Ebene Anwendung finden, weshalb diese womöglich noch lange Zeit ihre Daseinsberechtigung besitzen werden (Abbildung 3.7).

Jedoch ist dieser Wandel alleine für weitere Rechenleistungssteigerung ungenügend, sodass neue Speicher wie PCM, STT-MRAM oder Memristor sowie Packtechniken wie die 2.5D-

Integration oder das dreidimensionale Stapeln seitens Forschung und Industrie in Betracht gezogen werden. Diese zeigen bereits neue Dimensionen im Bereich der Speicherbandbreite, Speicherlatenz, Speicherkapazität, Zuverlässigkeit und Energieeffizienz auf. Am vielversprechendsten erscheint seit der Produktionsreife von TSVs das dreidimensionale vertikale Stapeln, kann doch gleichzeitig eine heterogene Integration sowie eine Reduktion des kritischen Pfads, der Datentransfers, des Energiekonsums und damit eine Effizienzsteigerung erzielt werden. Gerade die heterogene Integration von unterschiedlichen Materialien, Technologien und von Logik in den Speicher bietet das Potential, neuartige, datennahe Verarbeitungskonzepte zu entwerfen. Wiese könnten die aus der Kontaktstiftlimitierung resultierende *Bandwidth Wall* und die vorhandene *Memory Wall* überwinden, da externe Kommunikation minimiert oder gar vermieden werden kann (Unterabschnitt 2.1.11). Zeitgleich bieten Speicher wie der *Hybrid Memory Cube* eine paketbasierte Kommunikation an Stelle eines klassischen Speicherprotokolls, wodurch eine Abstraktion des internen Speicheraufbaus und somit vereinfachte Nutzung ermöglicht wird.

Die Verdrahtung von externen nichtflüchtigen Speicher-Chips ist zugleich Teil einer Speicherarchitektur, sodass Konzepte wie UMA, NUMA, CoMA sowie disjunkte und vereinte Speicher im Hinblick auf die Speichersicht der angeschlossenen Nutzer erörtert werden. Disjunkte und vereinte Speicher sind in der Domäne der Beschleunigerarchitekturen von hoher Aktualität, da aufgrund eines kohärenten Verhaltens keine Notwendigkeit mehr gegeben ist, Daten explizit zwischen verteilten Speichern zu kopieren. Entsprechend von Bedeutung ist die dreidimensionale Integration von Logik in den Speicher, welche nicht nur eine derartige vereinte Speichersicht ermöglicht, sondern schlicht durch den daraus resultierenden Aufbau Kopieren von Daten ganz zu vermeiden vermag.

Obgleich erst im weiteren Sinne für Speicherarchitekturen von Relevanz müssen Rechnerarchitekturen die Auswirkungen durch die sich stetig weitende Leistungsdivergenz zwischen Rechenkapazität und Speicher mit Techniken mildern. Neben der aufgezeigten Cachehierarchie werden zumeist viele Hardware-Threads, tiefe Lade-/Speicherwarteschlangen, Vektorregister, Vorabrufen (engl.: *prefetching*) oder sogenannte Scratchpads (direkt adressierbare Speicher eingebettet auf demselben Die) verwendet. Weiterhin in der Literatur erwähnte Verfahren sind der *Slide-And-Data Buffer* oder die *Slow-Lane Instruction Queue*, welche versuchen, ausstehende Ladeoperationen umzuordnen, wodurch aktuell lauffähige Instruktionen unter Wahrung von Rahmenbedingungen vorab ausgeführt werden können. Ähnliche Methoden wurden im Rahmen der Kilo-Instruktions-Mikroprozessoren aufgezeigt, konnten sich jedoch nicht durchsetzen.

Kapitel 4

Speicherintegrierte und speichernahe Rechenverarbeitung

Rechensysteme stoßen aufgrund verschiedener physikalischer Limitierungen an die Grenzen für zunehmenden Rechenleistungsgewinn (Kapitel 2). Einen großen Teil hierzu tragen offenkundig langsame Speicherarchitekturen bei, welche durch ihre Optimierung hin auf Kosten und Kapazität den signifikanten Bedarf an mehr Speicherbandbreite nicht versorgen können. Jedoch existiert innerhalb von flüchtigen Speichern – wie im vorigen Kapitel illustriert – eine nahezu ungebündelte Speicherbandbreite, welche bedingt durch dezentralen Aufbau von Rechensystemen und daraus resultierenden, langsamen externen Bussen nicht abgeschöpft werden kann. Mit dem Ziel Hemmnisse bei der Steigerung der Rechenleistung abzubauen, um Rechenleistung und Effizienzen zu steigern, wird die Verschiebung von Rechenkapazität näher an oder gar die Re-Integration dessen in Speicherarchitekturen äußerst attraktiv. Insbesondere, da dies gleichzeitig den *Von-Neumann-Engpass* beseitigen und energieineffiziente Datentransfers minimieren könnte. Jedoch ist unklar wie solch Architektur im Detail auszusehen hat, an welchen Stellen des Speichers eine derartige integrierbar ist oder aber wie eine zugehörige Cachehierarchie auszusehen hat.

Bereits in der Vergangenheit wurden planare DRAM-Strukturen mit eingebetteten Architekturen studiert, welche speichernahe (engl.: *near-memory processing*, NMP) sowie speicherintegrierte Rechenverarbeitung (engl.: *processing-in-memory*, PIM) ermöglichten [Siegl et al., 2016a]. Wohingegen das Forschungsinteresse an derartigen Architekturen abflaute, entfachte die TSV-Technologie diesen alten Ansatz wieder, da disjunkte Prozesstechnologien und Materialien nun vertikal integrierbar sind (Unterabschnitt 3.1.2). Neben TSVs befeuert die Neuausrichtung auf den datenzentrischen Rechensystementwurf (engl.: *data centric*), weg vom rechenzentrischen (engl.: *compute centric*), mit entsprechendem Fokus auf datennahe Verarbeitung (engl.: *near-data processing*, NDP) eine erneute Beurteilung von speicherintegrierten Architekturen. Entsprechend widmet sich dieses Kapitel diesem revolutionären, hochaktuellen Ansatz, um historische Entwicklungen sowie Konzepte von speichernahen und speicherintegrierten Rechenverarbeitung aufzuzeigen. Hierzu werden vielversprechende Architekturen erarbeitet, um Rückschlüsse auf architektonische Optimierungen, Integrationsort, Aufbau der Cachehierarchie sowie generelle Problemstellungen zu ziehen.

4.1 Taxonomie von datennaher Rechenverarbeitung

Obgleich die in Unterabschnitt 3.3.1 erläuterten Techniken bis zum heutigen Tag essentiell notwendig sind, ist es mit ihnen unmöglich, die *Memory Wall* zu überwinden, da das Hauptproblem der Kontaktstiftlimitierung (engl.: *pin*) und die daraus resultierende Bandbreitenbeschränkung nur ungeeignet adressiert werden kann. Neue Verarbeitungskonzepte sind notwendig, um Berechnungen näher an die Orte zu verlagern, an welchen Daten gespeichert sind oder Daten erzeugt werden. Dies führt zur Vermeidung von andernfalls nötigen externen, kleinen und potentiell gemultiplexten Bussen, welche mit niedrigen Frequenzen arbeiten, sowie tiefen Speicherhierarchien, welche eine große Menge an Datenbewegungen produzieren (von Festplatte bis zum Prozessorregister).

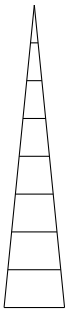
Im Jahr 1996 zweifelten bereits Saulsbury et al. die architektonische (Forschungs-) Ausrichtung in Richtung von prozessorzentrischen Verbesserungen an [Saulsbury et al., 1996]. Ein Jahr später zeigen Kaxiras et al. auf, wie der externe Datenverkehr unter Nutzung von Paketen (engl.: *parcels*) signifikant reduziert werden kann, indem Berechnungen durch Versand näher an die Daten gebracht werden und so auf fernen Rechenwerken ausgeführt werden, anstatt wie zuvor große Datenmengen zu den lokalen Rechenwerken verschoben zu werden [Kaxiras et al., 1997]. Dieses Verschieben von Verarbeitungen zur Laufzeit an den Ort, an welchem die zu verarbeitenden Daten gespeichert sind, wurde nicht nur von dem Rechensystem DIVA adaptiert [Kaxiras et al., 1997], sondern auch von den Rechnerarchitekturen HTMT und MIND/Gilgamesh genutzt. Hierfür stellten diese eine spezifische *parcel*-Hardwareunterstützung oder aber boten spezielle Speicheroperationen, welche den Speicherort vorab kannten [Brockman et al., 2004, Thoziyoor et al., 2005]. In heutigen Höchstleistungsrechnern wird hingegen zumeist auf Softwarelösungen zurückgegriffen, welche mit der Begrifflichkeit der *aktiven Nachrichten* (engl.: *active message*) beschrieben werden [Palumbo et al., 1992]. Dabei wird eine Nachricht an einen fernen, die Daten beinhaltenden Rechenknoten versendet, welcher daraufhin eine vordefinierte Softwarefunktion zu deren Manipulation derer ausführt.

Jüngst wurde von Kogge et al. eine neuartige Rechenmaschine mit dem Titel *Emu 1* vorgestellt, welche einen speicherzentrischen Architekturentwurf eines Hochleistungsrechners darstellt. Dabei wird auf dem Paradigma der datennahen Verarbeitung (NDP) aufgebaut; das Kernstück stellen leichtgewichtige *Threads* dar, welche durch dedizierte Hardwareeinheiten schnellstmöglich auf ferne Rechenknoten verschoben werden können [Minutoli et al., 2015, Dysart et al., 2016].

Es ist anzumerken, dass das Konzept der Aufgabenverschiebung vielversprechend für zukünftige datenzentrische Rechensysteme ist, jedoch kann rein durch den Aufgabenkontexttransfer (d.h. des Registersatzes) hin zu einem fernen Ort bereits ein erheblicher Mehraufwand entstehen. Beim Nachrichtenaustausch mit Hilfe sogenannter *aktiver Nachrichten* entfällt dies hingegen, da zumeist nur ein Funktionszeiger ausgetauscht werden muss, wohingegen der entfernte Rechner rein die Funktion aufrufen muss. Somit ist noch fraglich ob neu erscheinende Systeme, welche das alte Konzept des Thread-Verschiebens anstatt des aktuellen Nachrichtenansatzes verfolgen, profitabler sind.

Im Gegensatz zur einfachen Verschiebung von einzelnen Aufgaben zu fernen zu verarbeitenden Daten zeigten im Jahr 2001 Riedel et al. mit aktiven Festplatten auf, dass es neuartige Konzepte benötigt, um die durch Verwendung etablierter Techniken (Unterabschnitt 3.3.1) entstandene Lücke zwischen Rechenwerken und Daten zu schließen [Riedel et al., 2001]. IBM führ-

te jüngst diese philosophische Umstellung von prozessorzentrischer zu datenzentrischer Verarbeitung mit Hilfe ihrer datenzentrischen Rechensystemarchitektur (engl.: *data-centric systems*, DCS) fort, indem spezialisierte Rechenwerke über die vollständige Speicherhierarchie verteilt werden [Easton, 2013, Keable, 2012]. Agerwala führt hierzu aus, dass Datenübertragung außerordentlich aufwendig und somit teuer ist, sodass Berechnungen dort platziert werden sollen, wo immer sie am besten laufen [Agerwala, 2014]. Solch Anstrengungen, d.h. die Einbettung von



Rechnen mit Zwischenspeicher, d.h. regulärer Prozessor
Speichernahe Verarbeitung (engl.: *near-memory processing*, NMP) [Nair et al., 2015]
Berechnung im Speicher (engl.: *processing-in-memory*, PIM) [Elliott et al., 1992]
Rechnen in *Storage Class Memory* (*SCM*) [Agerwala, 2014]
Festplattennahe Verarbeitung [Davidson et al., 2006]
Rechnen in NAND *Flash* [Fitch et al., 2009, Keable, 2012]
Berechnungen innerhalb der Festplatte [Riedel et al., 2001]
Intelligente Netzwerke [Haring et al., 2012, Trader, 2015, Shainer, 2016]

Abb. 4.1: Konzept der datennahen Verarbeitung angewandt auf klassische Speicherhierarchie.

(heterogenen) Rechenwerken an jegliche Stelle an welcher Daten gespeichert werden, können als Verlagerung von einer statischen Speicherhierarchie mit (zentralem) Prozessor hin zu einer Hierarchie aus Rechenwerken bzw. einem hierarchischen, aktiven Datenspeicher zusammengefasst werden (Abbildung 4.1).

4.1.1 Datennahe Rechenverarbeitung (NDP)

Die datennahe Verarbeitung (engl.: *near-data processing*, NDP) errang eine große Aufmerksamkeit nicht nur aufgrund von IBMs Anstrengungen, sondern einerseits durch das Aufkommen des Marketingkonzepts *Big Data* [Gao et al., 2015] als auch andererseits durch die Forschung an datenzentrischen Rechenarchitekturen [Ranganathan, 2011]. Hinzu kommen die jüngsten technologischen Entwicklungen im Bereich der dreidimensionalen Integration initiiert durch aktuelle Forschung bezüglich Verarbeitung direkt im Speicher (PIM)[1]; diese verdankt ihre Fähigkeit der direkten Vernetzung durch Integration von maßgefertiger Logik in dreidimensional gestapelten Speicher (Unterabschnitt 4.1.2). Einige wenige kommerziell verfügbare und der NDP-Philosophie folgende Produkte existieren bereits. Hierunter fallen z.B. IBMs Netezza-Plattform [Davidson et al., 2006], in Auszügen IBMs BlueGene/Q (bei welcher die Aggregation im Netzwerk durch Fließkommaberechnung erweitert wurde) [Haring et al., 2012] sowie IBMs Produkt Hybrid Scalable Solid State Storage (HS4) [Keable, 2012, Fitch et al., 2009]. Ähnlich BlueGene/Qs Fließkommaberechnung im Netzwerk kündigte Mellanox ihre NDP-inspirierten Netzwerkswitchserie IB-2 im Jahr 2015 an, welche transparente Rechenfähigkeit innerhalb des Netzwerkswitches bietet und somit erlaubt, Daten die im Netzwerk den Switch passieren, instantan zu manipulieren [Shainer, 2016, Trader, 2015]. Mit der Übernahme der Firma EZchip und den mit Rechenkapazitäten ausgestatteten Netzwerkswitches und Netzwerkadaptern [Morgan, 2016], scheint sich Mellanox zu einem Vorreiter für die Schlüsseltechnologie der Verarbeitung im Netzwerk als auch der festplattennahen Berechnungen zu etablieren.

Da das Konzept der datennahen Verarbeitung keine Begrenzung seitens der allgemein be-

[1] Viele Forscher setzen PIM NDP gleich, obgleich PIM eine Untermenge von NDP darstellt (Abbildung 4.1).

kannten Speicherhierarchie widerfährt (Abbildung 4.1), schlussfolgert Agerwala, dass NDP in einer Vielzahl von Einsatzgebieten genutzt werden kann [Agerwala, 2014]. Jedoch brechen NDP-Rechnerarchitekturen mit Von-Neumann's Trennung (z.B. [Dlugosch et al., 2014]), wodurch die neuartigen Architekturen unausweichlich revolutionäre Änderungen an aktuell gegebene Programmierparadigmen stellen. In diesem Sinne schlägt Ranganathan vor, flüchtiges DRAM komplett mit nichtflüchtigem Speicher (engl.: *non-volatile memory*, NVM) zu ersetzen, da dieser signifikant mehr Speichervolumen als auch bedeutend geringeren Energiekonsum bietet [Ranganathan, 2011]. Als Resultat würden aktuelle Rechensysteme mit regulären Lade- und Speicheroperationen direkt nichtflüchtigen Speicher wie z.B. Flash adressieren. Im Detail beschreibt Ranganathan eine dreidimensional gestapelte Kombination aus *Phase-Change Memory* (*PCM*) und einem Memristor, welcher in Anbetracht von aktuell datenintensiven und teils NAND-Flash-Speicher-basierten Rechnersystemen eine wesentliche Architekturveränderung darstellt. Die Idee mag abwegig erscheinen, da nichtflüchtige Speicher im Gegensatz zu flüchtigen deutlich geringere Bandbreiten und höhere Latenzen bieten; hierbei ist jedoch zu beachten, dass bereits Seagates neu angekündigtes NVMe-basiertes Solid-State Festplatte (SSD) einen Durchsatz von 10 GB/s bietet [Seagate, 2016]. Entsprechend dringt dies in das Mittelklassesegment von flüchtigen Speichern vor (etwa einkanalige DDR3-1333 oder DDR4-1600). Hinzu kommt, dass derartige nichtflüchtigen Speicher einen dedizierten, jedoch transparenten und vom Hersteller gesperrten Prozessor bieten, wodurch bereits heutzutage das NDP-Konzept mit Rechenverarbeitungen im nichtflüchtigen NAND-Flash-Speicher möglich wäre.

Es existiert eine Vielzahl an datennahen Verarbeitungsideen als auch Konzepten (Abbildung 4.1); die namentlich bekannteste hiervon tätigt Rechenverarbeitung direkt im Speicher (engl.: *processing-in-memory*, PIM) und hat das Potential, die Auswirkungen der *Memory Wall* zu minimieren und somit die ExaFLOPS-Hürde zu überwinden. Die nachfolgenden Kapitel widmen sich daher explizit der PIM-Thematik und beleuchten die Vorzüge und Nachteile hiervon (Unterabschnitt 4.1.2).

4.1.2 Speicherintegrierte Rechenverarbeitung (PIM)

Eines der vielversprechenden Konzepte aus dem NDP-Kontext ist das Verarbeiten im Speicher – auch speicherintegrierte Verarbeitung (engl.: *processing-in-memory*, PIM) genannt – sowie die Klasse der datennahen Verarbeitung (engl.: *near-memory processing*, NMP). Beide versuchen, die aus der *Memory Wall*, der *Bandwidth Wall* als auch *Power Wall* entstandenen Lücken entweder ganz zu schließen oder zumindest zu reduzieren, indem Recheneinheiten näher an oder gar ganz in das Speichersubsystem eingebracht werden. Dabei wird jedoch nicht nur beabsichtigt, höhere Effizienz im Bereich der Verarbeitung und des Energieverbrauch zu erzielen, sondern explizit eine ungebündelte Speicherbandbreite als auch kürzere Speicherlatenzen zwischen Recheneinheit und Speicherbank zu generieren.

In Anbetracht des Aufbaus eines flüchtigen Speichers, als auch im Hinblick auf die dreidimensionale Stapelung dessen, ergibt sich eine Vielzahl von Stellen, an welchen Recheneinheiten eingebettet werden können. Diese Vielzahl wurde bereits in der Literatur in Betracht gezogen, sodass im Folgenden nur eine kleine Auswahl genannt werden soll:

- Zwischen dem Stromerfassungsverstärker (engl.: *sense amplifier*) und dem Spaltedekoder [Elliott et al., 1992, Kim et al., 1997]

- Hinter dem Spaltendekoder [Gokhale et al., 1995, Aimoto et al., 1996]
- In der Verarbeitungspipeline eines Prozessors integrierter Speicher [Thoziyoor et al., 2005].
- Integriert entweder vor oder hinter dem Speicherbus [Kogge, 1994, Shimizu et al., 1996, Patterson et al., 1997a], oder vor oder hinter dem im Speicher vorhandenen *Crossbar Switch* [Nair et al., 2015].
- Bei dreidimensionalen Speichern verteilt über jeden sogenannten *Vault* [Ahn et al., 2015]
- Bei dreidimensionalen Speichern als vollständige Ebene ausschließlich für Verarbeitungen eingebettetet [Zhu et al., 2013].

Da das Speichersubsystem aus einer Vielzahl von Speicherchips zusammengesetzt ist, jeder davon mit einer gewissen Menge an Speicherbänken und im dreidimensionalen Fall auch mit mehreren sogenannten *Vaults*, nutzt eine PIM-Architektur diesen Aufbau üblicherweise durch Parallelismus aus, indem multiple Recheneinheiten verteilt in den Speicher eingebracht werden. Hinzu kommt meist eine anwendungsspezifische Architektur, welche Energieeffizienz, Rechenleistung und dementsprechend den Gesamtrechendurchsatz signifikant verbessert. Entsprechend können die aus *Power Wall* und *Instruction-Level Parallelism* (ILP) *Wall* abgeleiteten Beschränkungen minimiert werden (solch Techniken werden in Unterabschnitt 3.3.1 beschrieben). Wie bei jedem Hardware-Entwurf ist auch bei PIM eine Balance nötig, resultierend aus der verfügbaren Fläche, den thermischen Zielen, den Kosten und der Zuverlässigkeit. Jedoch sind insbesondere die erstgenannten für PIM eine noch zentralere Problemstellung (Abschnitt 4.5).

Es fällt schwer, eine klare Abtrennung zwischen PIM und anderen, eng miteinander verwandten Architekturen zu ziehen. Beispielsweise teilt die Klasse der sogenannten *Processing Arrays* (z.B. 1973: Reddaways DAP [Reddaway, 1973], 2012: Adapteva Epiphany [Olofsson et al., 2014], 2015: REX Computing Neo [Hemsoth, 2015]) eine Vielzahl an Eigenschaften mit PIM, wohingegen sogar PIM wiederum *Processing Arrays* in einigen Studien selbst adaptiert [Gokhale et al., 1995]. Allerdings kann PIM im Allgemeinen die Eigenschaft einer passiven Beschleunigerarchitektur zugerechnet werden, welche im Standardmodus als transparente Einheit(en) in regulären Datenspeicher eingebettet sind. Sobald jedoch der zentrale Prozessor eine gewisse Aufgabe delegiert hat, können meist beliebige Instruktionen direkt auf Daten im Speicher angewandt werden.

Im zeitlichen Verlauf waren PIM-Architekturen wiederholt und immer dann von wissenschaftlicher Bedeutung, wenn neuartige Technologieaspekte oder Hürden (Unterabschnitt 2.1.10) erschienen. Im folgenden Abschnitt 4.2 wird ein historischer Überblick über PIM geboten, welcher gleichsam die architektonischen Entwicklungen im Bereich der datennahen Verarbeitungen umfasst.

Es soll nicht unerwähnt bleiben, dass PIM-Architekturen je nach Autor und zum Teil je nach Jahr der Publikation mit unterschiedlichen Akronymen betitelt wurden, obgleich sich die Grundidee nie änderte. Zu Beginn, als einfache Logik in die damals noch junge DRAM-Technologie eingebettet wurde, wurde PIM zumeist anhand der technischen Aspekte benannt, z.B. *Logic-in-Memory* (Minnick, Stone und Kautz) [Stone, 1970], *Merged Logic DRAM* (MLD) [Kang et al., 1999] oder *logic-embedded memories* [Kugler et al., 1996]. Ab dem Jahr 1990 erhielten PIM-Architekturen einen eher kognitiven Titel, etwa *Intelligent RAM* (IRAM, 1996) [Patterson et al., 1996, Kang et al., 1999], *Active Memory Model* (AMM, 1997) [Kogge et al., 1997] (2007: *Active Memory Operations* [Fang et al., 2007], 2012: *active memory controller* [Fang et al., 2012],

2015: *Active Memory Cube* (AMC) [Nair et al., 2015]) oder *smart memories* (1999, 2000) [Elliott et al., 1999, Mai et al., 2000].

4.2 Historische Betrachtung von speicherintegrierter Rechenverarbeitung

Die mit dem Entwurf des *EDVAC*-Computers [von Neumann, 1945] (Unterabschnitt 2.1.2) vor 70 Jahren von Von-Neumann konzipierte Trennung von Prozessoreinheit und Speicher, hilft nicht nur bei der Entwicklung der aktuellen Rechnerarchitektur, sondern auch der Steigerung in der Technologie: Prozessoren können mit großen Transistoren auf Rechenleistung hin optimiert, während Speicher mit kleinen, dicht pack-baren Transistoren auf große Kapazität und geringerem Energiekonsum abgestimmt werden [Saulsbury et al., 1996]. Obgleich dieser archi-

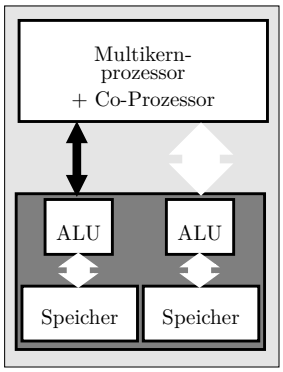


Abb. 4.2: Einbettung von dedizierten Recheneinheiten in regulären Speicher. Interne Recheneinheiten erhalten dadurch Zugriff auf eine signifikant höhere Speicherbandbreite.

tektonische Vorteil bis zum heutigen Tag erfolgreich ist, stellt das PIM-Konzept dieses durch die Re-Integration beider Komponenten auf demselben Chip (*Die*) oder eingebettet in einen gemeinsamen Stapelchip wieder in Frage (Abbildung 4.2) [Stone, 1970].

4.2.1 Zweidimensionale Speicherarchitekturen mit integrierten Recheneinheiten

Die Forschung zur Thematik, wie Recheneinheiten näher an die Daten und damit den Speicher herangebracht werden können, datiert bis in die 1960-er Jahre zurück. Es tauchte mehrmals in der geschichtlichen Entwicklung auf (Abbildung 4.3). Als Dennard die Ein-Transistor-DRAM-Zelle erfand [Dennard, 1968], beschrieben Minnick (1964: [Minnick, 1967]), Kautz (1969: [Kautz, 1969]) und Stone die zu der Zeit sogenannte *Processing Cellular Logic*, welche zusammen verschaltet zu hochparallelen Geflechten geformt werden konnte. Diese boten für heutige Verhältnisse winzige verteilte Speicher mit Rechenkapazität und ermöglichten damit, traversierende Daten instantan (engl.: *on-the-flight*) zu verarbeiten [Minnick et al., 1966]. Derartige Geflechte an *Cellular Logic-In-Memory* (CLIM) waren jedoch zumeist Spezialkonstruktionen mit festverdrahteter Funktionalität, sodass im Jahr 1970 Stone eine generische Lösung, den sogenannten “Logik in Speicher”-Prozessor vorschlug [Stone, 1970]. Es steht zur Debatte, ob diese CLIM-Strukturen bereits eine echte PIM-Architektur darstellen, da diese zumeist auf ein Anwendungsszenario

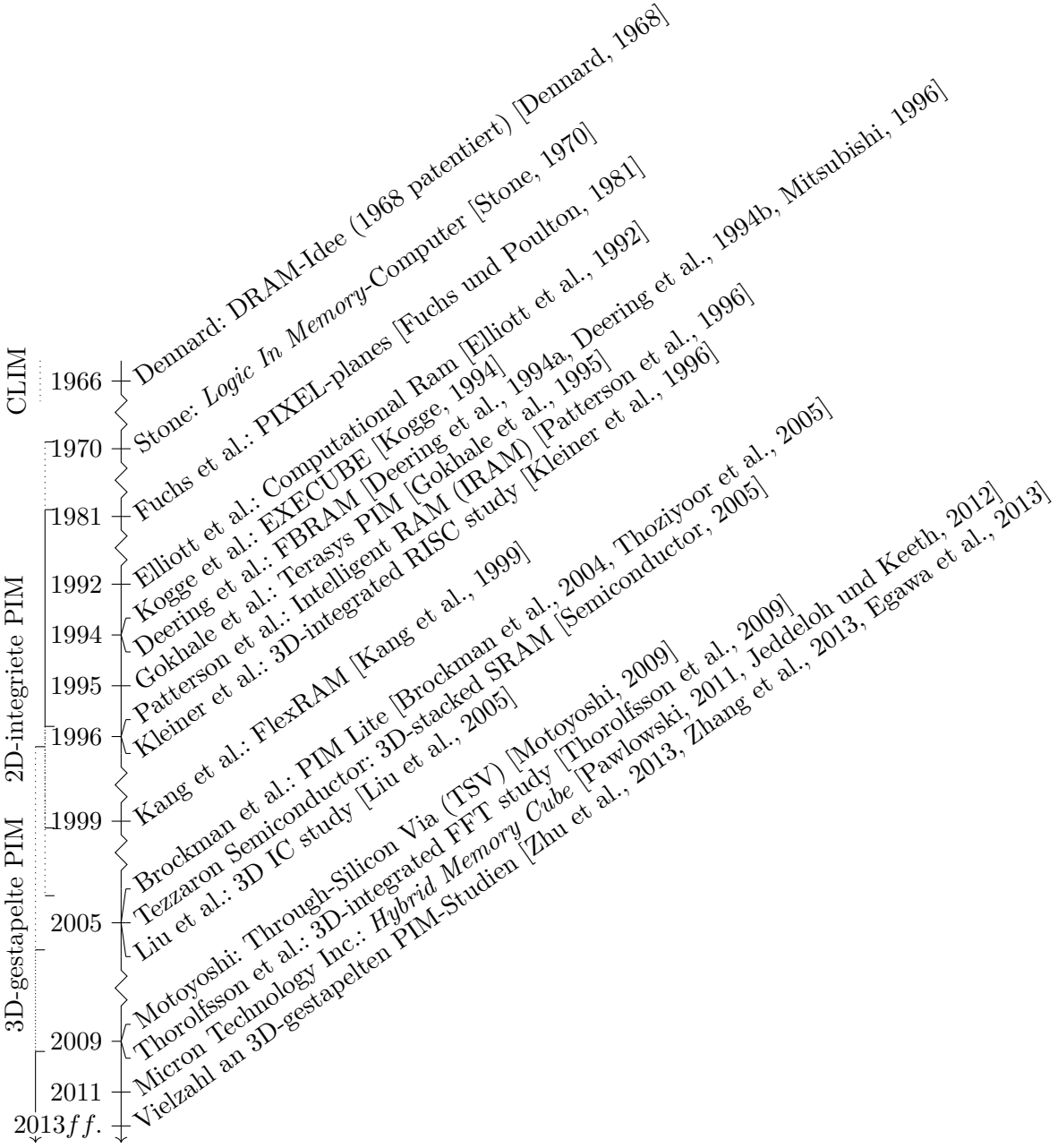


Abb. 4.3: Historische Entwicklungen zweidimensional und dreidimensional integrierter Schaltkreise im Bezug auf die PIM-Forschung.

limitiert waren. Jedoch kann die fundamentale Idee der Re-Integration von Rechenlogik und Speicher bei diesen primitiven Architekturen wahrgenommen werden.

PIMs Stärken, insbesondere die hohe Speicherbandbreite, führte Fuchs et al. etwa zehn Jahre später zum Entwurf der PIXEL-planes-Architektur. Diese umging die Speicherbandbreitenbeschränkung des Bildzwischenspeicher (engl.: *framebuffer*) von damaligen Grafikkarten, indem für jeden Pixel spezialisierte Rechenelemente in den DRAM-Speicher integriert wurden [Fuchs und Poulton, 1981]. Während die PIXEL-planes-Architektur ausschließlich für den graphischen Spezialzweck konstruiert wurde, entwarfen zehn Jahre später Elliott et al. einen stark parallelen Allzweckbeschleuniger, um die weiteren Vorteile von PIM herauszuarbeiten [Elliott et al., 1992]. Beide Rechnerarchitekturen können als erste "wahre" speicherintegrierte Allzweckbeschleuniger, entsprechend speicherintegrierte Rechenverarbeitung angesehen werden, da beide die typischen PIM-Charakteristiken wie kurze Latenzen in Verbindung mit hoher Speicherbandbreite und eine verbesserte Energieeffizienz aufweisen.

Obgleich alle genannten Architekturen bereits die essentiellen Merkmale von PIM aufweisen, Prozessor und Speicher sind eine eingebettete Konstruktion auf einem einzelnen Chip, so werden diese noch nicht als PIM bezeichnet. Der Begriff PIM wurde zuerst von dem namhaften PIM-Forscher Kogge im Zusammenhang mit seiner PIM-Architektur *EXECUBE* im Jahr 1995 genannt, wobei PIM dabei noch für *Processor In Memory* (deutsch: Prozessor im Speicher) abgekürzt wurde [Kogge et al., 1995, Kogge, 1994]. Die heute geläufige Form *Processing-In-Memory* (deutsch: speicherintegrierte Rechenverarbeitung) kann auf Gokhale zurückgeführt werden, welche diesen Begriff im Zusammenhang mit dem PIM *Terasys* nannte [Gokhale et al., 1995].

Die bis dato präsentierten PIM-Architekturen hatten entweder mit einem komplexen Speicherlayout, einer vordefinierten Logik oder einer schwierigen Softwareprogrammierung zu kämpfen. Im Kontrast dazu stellt Patterson et al. im Jahr 1996 einen MIPS-Einkernprozessor mit dem Namen *Vector Intelligent RAM* (VIRAM) vor, welcher dem PIM-Konzept folgend in den Speicher integriert war und diese Problematiken explizit adressierte [Patterson et al., 1996]. Zugleich unterscheidet sich der Ansatz von anderen Studien, welche auf eine Vielzahl an hochgradig parallelen Rechenelementen setzten und entweder zwischen den *Sense Amplifier* (deutsch: Stromerfassungsverstärkern) und dem Spaltendekoder [Gokhale et al., 1995, Elliott et al., 1992] oder direkt hinter dem Spaltendekoder [Kogge et al., 1995] eingebettet wurden. VIRAM erschien zur Hochphase der zweidimensional integrierten PIM-Forschung und zeigte auf, dass eine maßgeschneiderte Rechnerarchitektur für PIM zwingend nötig ist (in Falle von VIRAM: SIMD-Vektorisierung) [Patterson et al., 1997a]. Jedoch war VIRAM, wie alle anderen zweidimensionalen PIM, limitiert durch langsame Rechentransistoren, da eine (modifizierte) DRAM-Technologie mit großen Logikzellen genutzt wurde [Stone, 1970, Elliott et al., 1992, Gokhale et al., 1995, Saulsbury et al., 1996]. Hinzu kam im Allgemeinen die mangelhafte Möglichkeit der Aufrüstung von der Speicher- oder Rechenkapazität; hinzu kam die Problematik der nicht vorhandenen Vermarktungsidee. Obwohl die Forschungshochphase von zweidimensional integrierten PIM bereits beendet war, erschienen zwei weitere Studien: Im Jahr 1999 stellen Kang et al. *FlexRAM* vor, eine Architektur, welche einerseits Rechenelemente an jeder Speicherbank einbettete, um hohe Parallelität zu erzielen, und andererseits einen dedizierten starken Prozessor zum Verbund aus rechenaktivierten Speicherbänken hinzufügte, wodurch auch nicht hochgradig parallele Software davon profitierten konnte [Kang et al., 1999]. Ein weiterer Beitrag zur zweidimensional integrierten PIM-Forschung wurde sechs Jahre später von Brockman et al. erbracht, welche die in einer regulären Rechenarchitektur vorhandenen Zwischenregister mit großen Speichern erweiterten [Brockman et al.,

2004, Thoziyoor et al., 2005].

Zu den bereits bekannten Schwachstellen der aus den zweidimensional integrierten PIM-Forschung resultierenden Rechenarchitekturen, kam das junge Forschungsfeld GPGPU (engl.: *general-purpose computing on GPU*, GPGPU) hinzu und brachte zunächst die weitere Forschung an PIM zum Erliegen. Im Jahr 2001 wurde von Larsen et al. eine einfache Möglichkeit aufgezeigt die hochparallele Rechenleistung von sehr günstigen, vielfach verfügbaren und hinreichend energieeffizienten Grafikkarten zu nutzen [Larsen und McAllister, 2001]. Zusammenfassend schlussfolgern Saulsbury et al. und Patterson et al. jedoch, dass der Hauptgrund für den zweidimensional integrierten PIM-Misserfolg aus den zu hohen Kosten resultiert, welche durch die Ausbeute (engl.: *yield*), benötigter Metall-Lagen und die Einführung von Logik in die Speicherindustrie entstand. Gerade letztere ist optimiert auf Bytes/Wafer und lässt primär keinen Spielraum für eine Flächennutzung außerhalb von Speicherkapazität; zugleich resultierte aus der genutzte DRAM-Technologie eine langsame Gesamtrechengeschwindigkeit. Die Technologie war jedoch notwendig, um überhaupt Prozessor und Speicher auf einem Chip zu vereinen [Saulsbury et al., 1996, Patterson et al., 1997a, Patterson et al., 1997b]. Abgesehen von Einzelfällen wie VIRAM wurde generell die mangelhafte Programmierbarkeit bemängelt, die allgemeine Speicherkapazität, um Instruktionen und Daten zu halten, der erhöhte Aufwand in der Verifikation und der Mangel einer tatsächlichen Geschäftsidee [Patterson et al., 1997b].

Nichtsdestotrotz erbrachten Dlugosch et al. im Jahr 2014 eine Studie im Rahmen der zweidimensionalen Integration von Prozessor und Speicher. Hierbei wurde jedoch auf einfache Automaten zurückgegriffen, welche in DRAM-Speicher eingebettet wurden und somit signifikante Leistungszuwächse bei der Verarbeitung von regulären Ausdrücken im Gegensatz zu einem zentralen Prozessor boten [Dlugosch et al., 2014]. Vorausgegangen war eine ähnliche Studie der Firma Venray Technologies, welche im Jahr 2014 unter dem Namen *TOMI* ein DRAM-Module anbot, bei welchem jeder DRAM-Chip mit einem 8-Kern-Allzweckprozessor ausgestattet war [Venray Technology Ldt., 2014]. Gleichzeitig war ein Netzwerk auf dem Modul eingebettet, wodurch eine Kommunikation zwischen den verteilten DRAM-Chips ermöglicht wurde. Hinzu kommt, dass der Hersteller Intel eingebetteten DRAM (eDRAM) sowie den Prozessor auf einer einzelnen Siliziumverdrahtungslage (engl.: *silicon interposer*) verbindet, wie bei der Haswell-Architektur bereits geschehen [Hammarlund et al., 2014].

Werden die Anstrengungen für frühe zweidimensional integrierte PIM-Forschung zusammengefasst, können folgende Aspekte aufgezeigt werden:

+ Eine Einbettung von Speicher und Logik ist technisch und technologisch möglich, sodass zwischen beiden ein breiter Bus geformt werden kann.

+ Bedingt durch die breite Busverbindung kann eine hohe Speicherbandbreite in Kombination mit einer geringen Speicherlatenz erzielt werden.

+ Der Schlüssel zu extremer Rechenbeschleunigung stellt eine spezialisierte Rechenarchitektur dar. Nicht spezialisierte Rechenarchitekturen bieten durch den Ansatz nur beschränkten Mehrwert.

+ Ein PIM kann nicht nur als Rechenbeschleuniger, sondern kann auch als regulärer DRAM-Speicher im Rahmen eines zentralen Prozessors angesehen werden. Entsprechend kann eine Adaptierung vereinfacht werden.

– Für eine erfolgreiche Adaptierung stellt die Programmierbarkeit die größte Hürde dar. Dies insbesondere, da diese abhängig von der Speichersicht sowie der Verwendung derer ist (4.3). Solche Expertise war jedoch bei Speicherherstellern nicht gegeben [Patterson et al., 1997b].

– Bedingt durch die Integration von Logik und Speicher kann nur DRAM- oder Logik-Prozesstechnologie gewählt werden, wodurch zwischen langsamer Logik und hoher Speicherkapazität oder umgekehrt entschieden werden muss.

– Unverzichtbar ist eine klare Geschäftsidee, da PIM gegenüber regulärem Speicher bzgl. Bits pro Fläche sowie Platinenlagen und Kosten nicht konkurrenzfähig ist.

– Bei Aufrüstung von entweder Speicher- oder Rechenkapazität muss beides ausgetauscht werden.

4.2.2 Dreidimensional gestapelte Speicherarchitekturen mit integrierten Recheneinheiten

Während der Hochphase der zweidimensional integrierten PIM-Forschung (Abbildung 4.3) stapelten Kleiner et al. den Hauptspeichers über den L1- und L2-Cache, welche ihrerseits auf einen RISC-Prozessor gesetzt wurden [Kleiner et al., 1996]. Eine Vielzahl an sehr kleinen vertikalen Durchkontaktierungen ($\varnothing$ ∼2-5 μm) wurden als Verbindungen zwischen den angrenzenden Schichten genutzt. Unter Zuhilfenahme eines breiten Busses erzielte die so implementierte datennahe Rechnerarchitektur eine verbesserte und beschleunigte Speicherhierarchie, welche eine herausragende Gesamtrechenleistung erzielte und zugleich Kontaktstift- als auch PCB-Verdrahtungslimitierungen geschickt umging. Ausschließlich mit einem stärkeren Prozessor ausgestattet wiederholte neun Jahre später Liu et al. die Studie von Kleiner et al., um abermals aufzuzeigen, dass der Verbund aus Hauptspeicher, optimierter Cachehierarchie und Prozessor eine - nach Liu et al. definierte - 'nahezu perfekte' Rechenleistung erzielt [Liu et al., 2005].

Wohingegen Liu et al. keine thermischen Beeinflussungen in Betracht ziehen, gehen Kleiner et al. fälschlicher Weise davon aus, dass dreidimensionale Stapelung keine thermischen Probleme aufweist, was im Kontrast zu aktuellen Forschungspublikationen steht [Kim und Song, 2014, Khurshid und Lipasti, 2013]. Obgleich das technologische Fundament der dreidimensionalen Verdrahtungstechnik, im spezifischen das *Through-Silicon Via* (im Gegensatz zu weiteren Techniken wie *wire bonding*, *package on package*, *package in package*, 2.5-dimensionale Integration, ... [Davis et al., 2005, Sakuma et al., 2008]) bereits eine Produktionsreife im Jahr 2009 besaß [Motoyoshi, 2009], sodass eine dreidimensionale Stapelung hätte vollzogen werden können, so war die dreidimensionale Integration und Stapelung zur Erzielung von datennaher Verarbeitung laut dem Erfinder Motoyoshi nicht von allgemeinem Forschungsinteresse während der Jahre 2004 bis 2011. Motoyoshi weist hierbei explizit auf, dass thermische Belastung, Geschwindigkeit, Ausbeute, Fläche und das Vernetzen von TSVs bereits unter Kontrolle waren. Einzige Ausnahme während dieser Zeit war die Studie eines dreidimensional integrierten FFT-Beschleunigers von Thorolfsson et al., welcher eine signifikante Reduktion von Verdrahtungslänge und Energie erzielte [Thorolfsson et al., 2009].

Zusammenfassend änderte sich die Ausgangslage aufgrund der folgenden Aspekte zugunsten von dreidimensionaler Stapelung von Chips:

1. Im Jahr 2011 kündigte Micron das Produkt *Hybrid Memory Cube* (HMC) an, welches erstmalig eine Integration von Logik mit Speicher ermöglicht, basierend auf der Trennung der Chips und der Kontaktierung mithilfe von TSVs. Somit können schnelle Logiktransistoren in einer dedizierten Logikschicht sowie Speicherkapazität in einer dedizierten Speicherschicht übereinander gestapelt und direkt miteinander verdrahtet werden. Dies umgeht somit die Hürden von zweidimensionaler Integration und ermöglicht die speicherintegrierte Verarbeitung PIM [Pawlowski, 2011, Jeddeloh und Keeth, 2012].
2. Datenintensive *Big-Data*-Anwendungen verfolgen das Ziel, Daten direkt im Hauptspeicher zu verarbeiten (engl.: *in-memory computing*), anstatt diese meist von langsamen Festplatten(-Arrays) zu laden, um diese zu verarbeiten [Easton, 2013, Zhu et al., 2013]. In der Vergangenheit war dies nur schwer möglich, da Hauptspeicher zumeist zu klein waren. Heutige Rechensysteme bieten jedoch Hauptspeicher in Terabyte-Größe [Starke et al., 2015], wohingegen Forscher sogar nichtflüchtigen NAND-Flash-Speicher vorschlagen, um noch größere Kapazitäten zu ermöglichen [Ranganathan, 2011]. Werden nun die Berechnungen noch weiter an den Speicher herangebracht, so würden *Big-Data*-Anwendungen ebenfalls profitieren.
3. Die bis zum heutigen Tag aus der *Memory Wall* resultierende ungelöste Problematik des Energiekonsums und der daraus resultierende Zwang zu Effizienz und energiebewusstem Rechnen benötigt neue Ideen: aktuelle Verbesserungen zu Allzweckprozessorarchitekturen werden für die Zukunft insbesondere beim Bezwingen der ExaFLOPS-Hürde ungenügend sein [Kogge et al., 2010]. Brauchbare Lösungen, um *Exascale* anzugehen, erfordert u.a. eine Reduktion an Datentransfers durch Erbringung der Rechenleistung am Ort der Datenspeicherung [Martonosi, 2014]. Dies wird insbesondere vom Konzept der speicherintegrierten Verarbeitung adressiert. Eine zusätzliche Spezialisierung im Sinne der Ausrichtung in Richtung Beschleuniger ist profitabel und wird heutzutage bereits von der Industrie in verschiedenen Bereichen wie etwa Intel mit ihrer Vielkernprozessorarchitektur (engl.: *many integrated core*, MIC) oder Xilinx und Altera mit ihren programmierbaren (Logik-)Gatter-Anordnungen (engl.: *field programmable gate array*, FPGA) vollzogen.
4. Programmiermodelle (engl.: *application programming interface*, API) wie OpenMP (Stichwort: *target directive* [Dagum und Menon, 1998]) oder OpenACC haben sich seit den Tagen der zweidimensionalen Integration weiterentwickelt, wodurch diese transparente und nahtlose Ausnutzung von Beschleunigern bieten. Würden solche APIs auf speicherintegrierte Recheneinheiten angewandt werden, würde dies die in der Vergangenheit aufgezeigten Programmierproblematik entschärften.

Mit dem Jahr 2013 beginnend führten diese Aspekte zu einem signifikanten Anstieg an Forschungsstudien im Bereich der dreidimensional gestapelten PIM als auch datennahen Rechnerarchitekturen, welche zumeist die dedizierte Logiklagen des von Micron vertriebenen HMC nutzen [Zhu et al., 2013, Zhang et al., 2013, Egawa et al., 2013, Loh et al., 2013, Nair et al., 2015]. Hinzu kommt, dass Aly et al. aufzeigten, wie zukünftige dreidimensionale Stapel an Speicher und Logik aussehen könnte, indem sie unterschiedliche Materialien und Komponenten unter Nutzung verschiedener Technologien heterogen integrierten [Aly et al., 2015]. Rückführend auf die Resultate von Forschung an zweidimensional integrierten PIM vermieden diese neuen Studien das gemeinsame Einbetten von Rechenkapazität auf denselben Lagen wie Speicher. Im Hinblick auf die Terminologie beschreitet aktuelle Forschung eher den Weg der speichernahen Verarbeitung im Sinne, dass Rechenelemente näher an Speicher gebracht werden, im Gegensatz zu einer tatsächlichen Integration bei welcher Prozessor und Speicher auf der gleichen Lage einbegettet werden.

Trotzdem werden heutige Studien nach letzterem und nach speicherintegrierten Verarbeitung (PIM) benannt.

Die meisten Studien zu datennahen Rechnerarchitekturen konzentrieren sich auf Allzweckprozessorarchitekturen [Scrbak et al., 2015], die meist eine Vielzahl an leichtgewichtigen Prozessorkernen [Kogge, 1994, Kogge et al., 1995] im Kontrast zu großen, leistungsstarken Prozessorkernen nutzen [Saulsbury et al., 1996]. Weitere Forschung wurde im Bereich der anwendungsspezifischen Logik vollzogen, welches als primäre Forschungsausrichtung von Loh et al. empfohlen wird [Loh et al., 2013]. Typische Entwicklung im Bereich der daraus resultierenden PIM-Allzweckrechnerarchitekturen waren Studien zu leistungsoptimierten Caches oder gar zwischenspeicherlosen Architekturen [Kogge, 1994] in Verbindung mit Spezialfähigkeiten wie SIMD-Vektorregistern [Patterson et al., 1997a]. Obgleich das einfache Einbetten von Standard-Allzweckprozessoren, welche mit demselben Instruktionssatz wie der Hauptprozessor ausgestattet sind, eine weniger radikale sondern eher eine evolutionäre Forschungsausrichtung darstellt, so erlaubt dies doch die Ausführung von unmodifizierten Programmen auf dem entsprechenden PIM. In Anbetracht ansonsten schwieriger Migration von Software ermöglicht dies eine langsame Portierung und Aufteilung von Anwendungen, wodurch nicht mehr der Hauptprozessor sondern zukünftig der Recheneinheiten integrierte Speicher für die Berechnungen genutzt wird.

Im Kontrast zu den Studien rund um PIM-Allzweckrechnerarchitekturen wurde auch Forschung zu datennahen Rechnerarchitekturen betrieben, darunter GPGPU [Zhang et al., 2014], VLIW (engl.: *very long instruction word*, VLIW) [Nair et al., 2015], CGRA (engl.: *coarse-grained reconfigurable architectures*, CGRA) [Farmahini-Farahani et al., 2015] und einfache ("schlaue") PIM-angereicherte Instruktionen [Ahn et al., 2015]; dies geschah im Hinblick auf die aus in vorheriger Forschung an zweidimensional integrierten PIM-Architekturen gemachten Erfahrungen. Diese zeigten jeweils auf, dass speicherspezialisierte Architekturen ein hohes Potential für energieeffizientes Rechnen bieten. Von besonderem Interesse ist dabei die GPU-Studie von Zhang et al. [Zhang et al., 2014], da Grafikkarten mit ihrer großen Anzahl an Hardware-Threads Speicherlatenzen vernachlässigen und gerade deshalb eine bedeutend große Speicherbandbreite abrufen können. Ebenfalls von Interesse sind CGRA-Architekturen, da diese das übliche Laden von Instruktionen und folglich den Bandbreitenmehraufwand und den Dekodierungsaufwand aufgrund des Prinzips einsparen, sodass in Folge ein geringerer Energiekonsum mit höherer Gesamtrechenleistung erzielbar ist.

Trotz der großen Zahl an PIM-Studien gibt es keine Konvention, welcher Typ von spezifischer Rechnerarchitektur für den Verbund aus Speicher und Rechenlogik am Besten geeignet ist, sodass rein die bekannten Limitierungen aus Flächenkonsum, Stromversorgung und Energieverlust, Kühlung und thermischen Beschränkungen gegeben sind [Kogge et al., 1995] (Abbildung 4.4). Als zusammenfassende Notiz zeigen Ranganathan and Balasubramonian et al. auf, dass in Bezug auf die Computerarchitekturen für speicherintegrierte Verarbeitungsarchitekturen alle alten Ideen wieder neu gedacht werden müssen [Ranganathan, 2011, Balasubramonian et al., 2014]. Zugleich adressieren Loh et al. den Rahmen der potentiellen Architekturen und deren Grenzen und klassifizieren sie mit den Begriffen *Compound PIM Operations* (CPO), d.h. den Turingfähigen Architekturen, und *Bounded-operand PIM Operations* (BPO), d.h. die an eine Funktion gebundene Architekturen ein [Loh et al., 2013].

Zusammenfassend zeigt die Forschung an dreidimensional gestapelter, datennaher Verarbeitung folgende Aspekte auf:

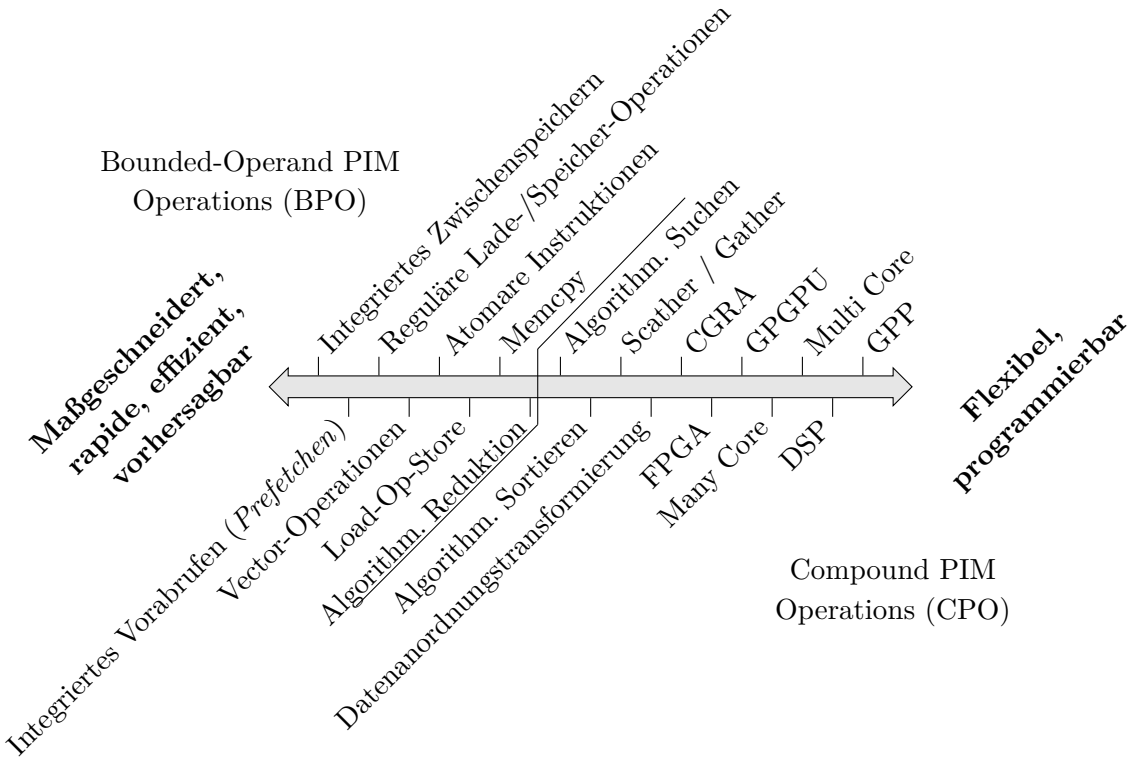


Abb. 4.4: Eine Taxonomie an potentieller PIM-Architekturen im Zusammenhang mit regulären Rechenarchitekturfähigkeiten [Loh et al., 2013]. Während BPO eine starre vordefinierte, nicht schleifenfähige Funktion ausführt, sind CPO zum großen Teil Turing-vollständig.

+ Die Stapelung von mehreren Lagen mit Hilfe von TSVs ermöglicht eine heterogene Integration von verschiedensten Technologien (Digital, Analog, RF, ...) und Materialien [Liu et al., 2005, Balasubramonian et al., 2014]. Zugleich kann die Energieeffizienz signifikant verbessert werden (Toshiba erzielt eine Verdopplung [Toshiba, 2017]).

+ Aufgrund der Flexibilität an Durchkontaktierungen kann die interne Busbreite angepasst und dadurch Speicherbandbreite individuell parallel angeboten werden. Da keine externe Busse und Kontaktstifte notwendig sind, resultieren höhere Frequenzen und geringere, elektrische Kapazitäten wodurch gleichzeitig eine Reduktion an Treibern und I/O Schnittstellen von statten gehen kann [Liu et al., 2005].

+ Der kritische Pfad kann aufgrund von kürzeren Verbindungen, und damit Gesamtlatenzen reduziert werden [Liu et al., 2005, Reddaway, 1973].

+ Höhere Effizienzen sowie geringerer Energieverbrauch sind durch geringere Latenzen, höhere Speicherbandbreite, geringerer externer Kommunikation sowie reduzierten Datentransfers zu erwarten [Patterson et al., 1997a].

+ Eine höher taktbare sowie effizientere Speicheransteuerung (engl.: *memory controller*) kann durch die Separation von Logik und Speicher realisiert werden.

+ Die Vielzahl verfügbarer externer Kontaktstiften ermöglicht weitere Einsatzzwecke wie z.B. die Integration von Netzwerkfähigkeiten im Gegensatz zu reinen Lade- und Speicheroperationen.

– Größter Mangel ist die Programmierbarkeit, welche durch die Speichersicht und einer eventuell vorhandenen MMU abhängig ist (Abschnitt 4.3).

– Es besteht keine individuelle Möglichkeit der Aufrüstung von entweder Logik oder Speicher.

– Durch dreidimensionale Stapelung zu erwarten sind ein erhöhter Test- und Validierungsaufwand, gestiegene Herstellungskosten als auch eine der komplexeren Technologie geschuldeten geringeren Ausbeute; insbesondere müssen Stapelung und jede Lage einzeln betrachtet werden [Sakuma et al., 2008].

+ Trotz intensiver Forschung existieren nur Ansätze jedoch keine klaren PIM-Architekturen.

4.2.3 Fallbeispiel: Der IBM Active Memory Cube (AMC)

In den Jahren 2011 bis 2015 forschte die Firma IBM im Rahmen des Forschungsprojekts *FastForward* an einer PIM-Architektur, welche als *Active Memory Cube* (AMC) tituliert wurde[2] [Nair et al., 2015]. Hauptanliegen war der Entwurf einer speicherintegrierten Beschleunigerarchitektur, welche die ExaFLOPS-Hürde unter den Beschränkungen der durch das Energieministerium der USA gesetzten Leistung von maximal 20 Megawatt erreichen konnte. Da eine einfache Verbreiterung der SIMD-Befehle in Kombination mit höherer Taktrate des vorausgehenden, eingebetteten IBM A2-Prozessors (IBM BlueGene/Q) die 20 Megawatt überschritten hätte, wurde auf die Integration einer dedizierten Beschleunigerarchitektur in einem dreidimensional gestapelten HMC-Speicher gesetzt mit einem IBM-POWER-Prozessor als zentrale Steuerung.

[2] Der Autor war in den Jahren 2012 bis 2014 Teil des Simulationsteams – bestehend aus insgesamt zwei Personen für die Logikebene und einer Person für den HMC-Speicher –, welches das taktgenaue AMC-Simulationsmodell entwickelte.

Die so vorgesehene speicherintegrierte Beschleunigerarchitektur sollte aus acht von Micron gefertigten Speicherlagen mit einer Gesamtspeicherkapazität von 8 GB und andererseits aus einer maßgeschneiderten Logiklage bestehen. Letztere erhielt einen für den AMC eigens entworfenen Netzwerkswitch, welcher die *Vaults* mit den bis zu 32 in der Logikebene integrierten VLIW-Beschleunigerkernen mit dem Hauptprozessor sowie anderen AMC verband. Der Netzwerkswitch wurde explizit auf einen hohen Datendurchsatz zwischen je einem VLIW-Beschleunigerkern und seinem zugeordneten *Vault* hin optimiert; jedoch kann jeder Kern innerhalb von HMC mit jedem *Vault* kommunizieren. Dabei sollte die maximal zu erzielende Speicherbandbreite 320 GB/s betragen. Die maximale Taktfrequenz für den kompletten Speicherstapel hingegen betrug 1,25 GHz. Ausgehend von einem signifikanten Bedarf an hoher Energieeffizienz wurde die Beschleunigerarchitektur trotz gegensätzlicher Ausrichtungen auf äußerst niedrigen Energieverbrauch und zugleich hohe Speicherbandbreitensaturierung getrimmt. Hierzu wurden mehrere Verfahren eingesetzt:

Um den Datentransfer bei Zwischenspeichern nebst korrespondierendem Energiekonsum zu vermeiden, wurden zwischen den Beschleunigerkernen und dem Speicherstapel keine Caches integriert. Die daraus resultierende Latenz wird einerseits durch den Einsatz einer 192 Elemente tiefen Warteschlange für ausstehende Lade-/Speicheroperationen je Beschleunigerkern gelindert. Zugleich ist jedes der sechzehn Vektorregister 32 Elemente tief (nicht breit; Stichwort: *SIMD-in-space*), wodurch mehr ausstehende Ladeoperationen je Register und entsprechend eine höhere Speicherbandbreitensättigung ermöglicht wird. Ein weiterer Aspekt die Warteschlange betreffend stellt eine Aggregation von Schreibzugriffen dar, welche zur Beschleunigung zusammengefasst werden können, sofern mehrere die selbige selbige 128 Byte Cachezeile adressieren. Zur Saturierung der tiefen Warteschlange waren vier Lade-/Speichereinheiten vorgesehen, welche parallel und ohne blockieren Anfragen stellen können.

Den Einfluss von *Interrupts* und *Traps* minimiert eine dedizierte MMU (IBM Nomenklatur: *effective to real address translation*, ERAT) mit wenigen Einträgen; diese wird entweder bei *Traps* durch das Betriebssystem IBM CNK oder im Falle des Höchstleistungsrechnens explizit vom Nutzer vor einer Berechnung aufgesetzt. Hinzu kommt die Vermeidung von instruktionsbedingter Ladeinterferenz durch ein in jeden Rechenbeschleunigerkern integriertes dediziertes *Scratchpad Memory* (deutsch: “Notizblockspeicher”). Dieser kann bis zu 512 Instruktionen halten und wird durch den Hauptprozessor vor Beginn der Ausführung mit einem Quellprogramm, meist einer einfachen Funktion (engl.: *kernel*), geladen. Diese Entwurfsentscheidung ist ein Zugeständnis an das Höchstleistungsrechnen, da somit die Rechengeschwindigkeit zwar erhöht, gleichzeitig jedoch mehr Siliziumfläche benötigt wird wodurch ein leicht höherer Energiekonsum resultiert.

Zu guter Letzt wurde eine exponierte Verarbeitungspipeline (engl.: *exposed pipeline*) entworfen, welche den Energieverbrauch durch den Wegfall von typischen Architekturkonzepten wie Registerumbenennung, die Ausführung in anderer Reihenfolge (engl.: *out-of-order execution*), die Sprungvorhersage sowie das klassische Verzögern aufgrund von Datenabhängigkeiten zur Verhinderung von Konflikten senkt. Einzig die Datenabhängigkeit hinsichtlich ausstehender Ladeoperationen wird in Hardware behandelt, da diese nicht vorhersagbar ist; die anderen Abhängigkeiten müssen durch den Programmierer oder Compiler explizit anhand der Länge der Verarbeitungspipeline beachtet werden. Zugleich waren zwei Verarbeitungslatenzen jeweils für Ganzzahl- sowie Fließkommaberechnungen gegeben, welche ebenfalls durch den Quelltext zu beachten sind. Somit ist bei schwer datenabhängigem Quelltext die Möglichkeit gegeben, Adressberechnungen explizit zwischen Fließkommaberechnungen zu verstecken, wodurch die Verarbeitungspipeline effizienter

genutzt werden kann. Sprungvorhersagen werden durch absolute Sprünge sowie dem Vorabrufen der kommenden Sprungoperation unnötig, da somit innerhalb eines Taktes die kommende auszuführende Instruktion vorab bekannt ist.

Obgleich die Architektur die Zielsetzung Energieeffizienz sowie Rechenleistung erfüllt, so wurde sie von den Forschungsergebnissen hinsichtlich zweidimensionaler PIM negativ betroffen. Gerade die Programmierbarkeit erwies sich bedingt durch den neu entwickelten Instruktionssatz für den Programmierer, jedoch insbesondere für den Compiler, als schwierig. Auch das Verteilen von Lade- und Speicheroperationen zur Minimierung von Speicherlatenzen gestaltete sich als Herausforderung, zumal die Architektur bei Ladeabhängigkeiten die Bearbeitung einer vollständigen VLIW-Instruktion anhalten muss. Demzufolge wurden von neun Instruktionen einer VLIW-Instruktion (entsprechend direkt adressierter Ausführungseinheiten), bei welchen eine einzige Abhängigkeit noch nicht aufgelöst war, alle ausführenden Einheiten von der weiteren Abarbeitung abgehalten. Hinzu kam ein hoher Preis resultierend aus der Integration von IBMs Logikebene und Microns Speicherebenen.

Trotzdem zeigten Nair et al. anhand des AMC, dass signifikante Zuwächse an Rechengeschwindigkeit in Kombination mit Reduzierung des Energieverbrauchs samt Erhöhung der Energieeffizienz erzielt werden können, indem solch maßgeschneiderte Recheneinheiten näher oder gar ganz in den Speicher integriert werden [Nair et al., 2015]. Dies konnte anhand der beiden aus dem *High Performance Linpack*-Benchmark extrahierten, synthetischen Anwendungen *DAXPY* ($y \leftarrow \alpha x + y$) und *DGEMM* ($C \leftarrow \alpha AB + \beta C$) aufgezeigt werden. Im Detail konnte *DGEMM* 83 % der verfügbaren Fließkommarechenleistung des AMC-Rechenbeschleuniger abrufen, was auf dessen hohe Lokalität zurückzuführen ist. Wird ein voll ausgestatteter Rechenknoten, also sechzehn AMC mit zentralem IBM-POWER-Prozessor angenommen, so würde dieser eine theoretische Leistung von 4,3 TFLOPS erzielen. Die Effizienz beträgt dabei 26 GFLOPS/Watt, welche im starken Kontrast steht zu Höchstleistungsrechensystemen wie dem IBM BlueGene/Q mit 2,1 GFLOPS/Watt oder Tsubname mit 4,4 GFLOPS/Watt.

Unabhängig davon konnten Baumeister et al. Energieeffizienz und Rechenleistung anhand der beiden Anwendungen *LBM* und *LQCD* eruieren [Baumeister et al., 2015]. Dabei zeigen Sie auf, dass der Instruktionssatz, die tiefen Vektorregister, der Instruktionsspeicher sowie die Speicherbandbreite und Speicherlatenz für beide Anwendungen von Vorteil sind. Gerade bei der Anwendung *LQCD* konnte eine Fließkommaeffizienz von 94 % erzielt werden. Hinsichtlich der hohen verfügbaren Speicherbandbreite konnten Baumeister et al. außerdem aufzeigen, dass beide Anwendungen ausschließlich von der Rechengeschwindigkeit abhängig sind. Bedingt durch den geringen Energiekonsum von 10 Watt pro AMC und der reinen Limitierung durch die Rechengeschwindigkeit erzielt ein AMC im Vergleich zu NVIDIAs *K20x*-Architektur eine Verbesserung des Energieverbrauchs für die Anwendung *LBM* um den Faktor 8,7 (im Detail: 0,3 μJ zu 2,6 μJ). Aufgrund dieser Ergebnisse und der hochbrisanten Forschung im Rahmen von PIM-Architekturen sowie dem Wandel hin zur Integration von Recheneinheiten in Speicher erhielten die Forscher daraufhin im Jahr 2015 den renommierten “Hans Meuer Award” der ISC International Supercomputing Conference [Pleiter, 2015].

4.3 PIM: Herausforderungen hinsichtlich der Verwendbarkeit

Die Konzepte datennahe Verarbeitung NDP und speicherintegrierte Verarbeitung PIM erzwingen nicht nur eine signifikante Veränderung im Entwurf von Rechnerarchitekturen sondern gleichzeitig durch den Bruch mit der klassischen *Von-Neumann-Architektur* ein Umdenken, wie solch hochparallele Beschleunigung effizient genutzt werden kann. Abgesehen von den bereits herausgearbeiteten Vorzügen seitens PIM bieten die meisten PIM in ihrem Standardmodus primär eine transparente Speicherkapazität, die wie regulärer DRAM-Speicher per Lade- und Speicheroperationen ausgehend von einem Hauptprozessor angesprochen werden kann. Dies führt zu einer einfachen, schnellen und nahtlosen Integration in heutige Rechnerarchitekturen und ermöglicht einen Datenaustausch zwischen PIM und Hauptprozessor, bei welchem kein Kopieren von Daten nötig ist [Elliott et al., 1992]. Zugleich fördert dies eine graduelle Ersetzung regulärer Speichermodule mit Modulen, die derartige Recheneinheiten bieten [Kim et al., 1997]. Obgleich in der Literatur davon abgesehen wird und typische Beschleuniger (mit Ausnahme von Intels MIC) dies ebenfalls nicht anwenden, könnte ein gemeinsamer Instruktionssatz angewandt werden, welcher von Hauptprozessor und Beschleuniger geteilt wird. Aktuelle Studien formen hierzu unter Zuhilfenahme von multiplen, einfachen ARM-Prozessorkernen einen hochgradig parallelen PIM [Menon et al., 2014], jedoch weisen Studien zweidimensional integrierter Architekturen wie Pattersons *IRAM* darauf hin, dass PIM nur mäßig von Allzweckprozessoren profitiert [Patterson et al., 1997a].

PIM können, wie in Unterabschnitt 4.1.2 herausgearbeitet, an unterschiedlichsten Stellen in einer Speicherarchitektur eingebettet werden, etwa integriert hinter oder vor einem Speicherbus, einem in den Speicher integrierten Switch (wie in einem HMC vorhanden), innerhalb einer Bank oder an Bänken, innerhalb eines dreidimensionalen Stapels oder über mehrere Speicherchips verteilt. Dies führt unausweichlich zu verschiedensten Speichersichten (im Hinblick auf Adressübersetzung), multiplen Ausführungsmodellen (SIMD, MIMD, ...), ungleichen Speicherlatenzen und resultiert infolgedessen in unterschiedlichen Anforderungen an Softwareprogrammiermodelle. Gerade letzteres stellt ein Hauptanliegen der Forschung im PIM-Bereich dar [Kogge et al., 1997], da bereits zweidimensional integrierte Rechnerarchitekturen daran scheiterten. Ein PIM kann als hochgradig paralleler Hauptprozessor oder als sehr schneller im Speicher integrierter Coprozessor genutzt werden, welcher entsprechende Aufgaben als SIMD oder MIMD ausführt [Kogge, 1994]. Jedoch ist ein klarer Trend in Richtung von SIMD feststellbar, da MIMD für die PIM-Recheneinheiten bedeuten würde, dass eine Vielzahl an Logik für das Einholen von Instruktionen und deren Dekodierung benötigt würde. Da die Fläche im Speicher jedoch hoch limitiert ist, würde dies zu einer geringeren Speicherkapazität führen. Zugleich ist angedacht, dass ein PIM die Rolle des aktiven Speichermanagements übernimmt, so dass einfache Aufgaben wie das Sammeln und Verteilen (engl.: *gather and scather*) von Daten an den PIM delegiert werden können. Daten würden somit zusammenhängend aus der Speicherhierarchie bezogen werden können, sodass die Speicherlatenz zeitgleich optimiert wird. Zusätzlich bietet ein heterogenes System aus PIMs und dediziertem Hauptprozessor eine Möglichkeit, letzteren in einen Schlafmodus zu setzen; während der Berechnungsphase der PIMs, führt dies zu einer höheren Gesamtsystemeffizienz [Shimizu et al., 1996].

Abgesehen vom Ausführungsmodell stellt eine weitere Schwierigkeit die Speicherverwaltungseinheit (engl.: *memory management unit*, MMU) dar, da Speicher im Normalfall physikalisch adressiert wird, während Software auf virtuellen Adressen arbeitet. Im Falle einer datennahen

Verarbeitung bedeutet dies, dass eine PIM-Architektur zumindest Auszüge aus einer global, zentral verwalteten MMU benötigt, um solch Adressübersetzung zu ermöglichen. Alternativ schlägt Ahn et al. vor, den PIM ausschließlich auf physikalischen Adressen agieren zu lassen, jedoch dessen Speichersicht auf Segmente, Seiten oder gar Cachezeilen zu limitieren [Reddaway, 1973, Ahn et al., 2015]. Einerseits hätte dies den Vorteil, dass durch die nun fehlende Adressübersetzung die Hardware vereinfacht und energieeffizienter würde, andererseits ein explizit durch den Hauptprozessor durchgeführter Datentransfer erbracht werden müsste, sobald PIMs untereinander Daten austauschen wollten. Während solch eingeschränkte Speichersicht einen klaren Einschnitt darstellt, das Teilen der MMU des Hauptprozessors höheren Druck aufbaut und im schlimmsten Fall Lade- und Speicheroperationen verlangsamt, würde eine dedizierte MMU je PIM in mehr Logik im Speicher resultieren. Zugleich würde eine dedizierte MMU einen Austausch von Einträgen aus der MMU zwischen Haupt- und PIM-MMU erfordern, welches in einem komplexeren Betriebssystem mündet. In Anbetracht von IBMs AMC (Unterabschnitt 4.2.3) und ähnlichen Studien scheint es, dass zukünftige PIM eine dedizierte MMU benötigen, da ein Einsatz als verteilte, hochparallele Rechenwerke vorgesehen ist, welche parallel riesige Datenmengen verarbeiten sollen.

Trotz Optimierungen an Speichersicht und Ausführungsmodell werden PIM wohl nicht auf expliziten Datentransfer zur Kommunikation verzichten können, da bereits heutige Speichermodule aus mehreren Chips bestehen. Solange auf den etablierten Speicherbussen für zukünftige PIM aufgesetzt wird, würde dies unausweichlich eine Anforderung an das Programmiermodell stellen, welches derartigen Datenaustausch transparent zur Verfügung stellen oder zumindest ermöglichen müsste. Zugleich wäre eine Aufgabenmigration wie in Abschnitt 4.1 beschrieben von Interesse. Eine Linderung dieser Problematik stellt Microns HMC dar, welcher nicht nur die Speichermodule stapelt, sondern zugleich – basierend auf dem Konzept von Netzwerkprotokollen – ein Netzwerk aus gestapelten Speichern bildet, wodurch eine Kommunikation ohne Hauptprozessor ermöglicht wird.

Ein Nachteil moderner Speicheransteuerung für PIM stellt das Multiplexen von Adressen dar. Dies wird zur Steigerung der Bandbreite und zur Minimierung der Latenz getätigt, wird jedoch meist in der prozessorintegrierten Speicheransteuerung getätigt, sodass die Speichersicht innerhalb eines PIMs nicht linear ausfällt. Jedoch kann das Multiplexen auch innerhalb der Speicheransteuerung innerhalb des Speichers getätigt werden, jedoch bedeutet dies einen Speicherbandbreitenverlust seitens des Prozessors. Typische PIM-bezogene Forschungsstudien betrachten diese Technik nicht im Zusammenhang: Sie konzentrieren sich zumeist ausschließlich auf die PIM-Rechenleistung sowie die zu erreichende Bandbreite, nicht jedoch auf die Bandbreitenverluste und Latenzauswüchse, die der zentrale Hauptprozessor erfährt. Hinzu kommt, dass bei aus PIM und Hauptprozessor mit klassischer Cachehierarchie ausgestatteten Systeme die Kohärenz und Konsistenz kritisch sind, da bereits im Cache vorgehaltene Daten innerhalb des Speichers durch den PIM manipuliert werden können, dies jedoch zurück an die Cachehierarchie propagiert werden muss.

Bedingt dadurch, dass es prinzipiell keine Einschränkung gibt, an welcher Stelle in einem Speicher oder innerhalb der vollständigen Speicherhierarchie eine Einbettung einer dedizierten Rechnerarchitektur vorgenommen werden kann [Kang et al., 1999], bietet dies eine offene Fragestellung inwiefern eine Anwendung eine solch speicherintegrierte und datennahe Rechnerarchitektur ausnutzen kann. Wohingegen nicht Turing-mächtige PIM, mit Hilfe von Register welche in den Adressbereich von Ein-/Ausgabe-Speicher abgebildet wurden [Gokhale et al., 1995] eventuell mit

einfachen Kommandos instruiert werden können, benötigen Turing-fähige PIM unter Umständen bereits eine neuartige Programmiersprache oder werden gar ausschließlich per Softwarebibliotheken aufgerufen, sodass Anwender diese transparent nutzen. Wohingegen im Hochleistungsrechnen der Aufwand vertretbar ist, eine Anwendung per Hand auf PIM zu optimieren, kann solch Ansatz wiederum nur schwer in schnelllebigen Umgebungen praktiziert werden. Die Erkenntnis, dass der Bereich von in DRAM-Speicher integrierten Recheneinheiten eine höhere Programmiersprache benötigt, erkannte bereits Stone im Jahr 1970, welcher jedoch DRAM ausschließlich als Cache kannte [Stone, 1970]. Hinzu kam, dass kein einfaches Programmierparadigma existierte, welches heterogene Beschleuniger durch transparente Arbeitsauslagerung ausnutzen konnte. Bedingt durch die heutige einfache Verfügbarkeit von Beschleunigern wie Vielkernprozessoren, GPU oder aber FPGA ist bereits eine signifikante Forschung in die Thematik der transparenten und nahtlosen Arbeitsauslagerung eingebracht worden, um die Rechengeschwindigkeit zu steigern und dementsprechend eine höhere Effizienz zu erhalten [Siegl et al., 2015]. Zur Adaption für PIM könnten explizit Beschleuniger adressierende Programmierumgebungen wie CUDA oder OpenCL von Interesse sein; ebenfalls denkbar ist der Einsatz der eher allgemeinen Programmierumgebungen für parallele Systeme wie OpenMP und OpenACC. Eine weitere Option wäre das heterogene Verarbeitungskonzept von AMD mit dem Titel *Heterogeneous System Architecture* (HSA) [Kyriazis, 2012].

Kern des HSA-Konzepts ist eine gemeinsame, virtuelle Speichersicht zwischen Host-CPU und Beschleuniger. Ein virtueller Instruktionssatz der sogenannte *Intermediate Layer* verhilft die ISA der CPU sowie des Beschleunigers zu abstrahieren um bei Bedarf den Quelltext entweder vorab bei der Kompilierung oder zur Laufzeit auf die tatsächliche ISA zu übersetzen. Hinzu kommt eine eigene Laufzeitumgebung, bei welcher zuvor annotierte, parallele Anteile einer Anwendung auf verfügbare Rechenelemente terminiert werden. Da HSA als Fundament für weitere APIs und Programmierumgebungen vorgesehen ist, können OpenCL, OpenMP oder weitere domänenspezifische Sprachen direkt auf HSA aufbauen. In Verbindung mit der Analyse von potentiellen Programmierparadigmata, ist eine Evaluierung der Hardware nötig, um einerseits eine effiziente Kommunikation zwischen mehreren verteilten PIM zu ermöglichen, andererseits um die Zwischenspeicherhierarchie kohärent und konsistent mit dem Hauptspeicher zu halten, welcher durch PIM modifiziert werden kann. Dies ist besonders bedeutsam, da Softwareprogrammierer i.a. von einer flachen, globalen Speichersicht ausgehen, welche sich nur unter Zutun ihrer eigenen Software verändert.

Mit unterschiedlichen Speichersichten und uneinheitlichen Speicherzugriffen (siehe NUMA-Architektur, Abschnitt 3.2) werden variierende Speicherlatenzen die Standardsituation für datennahe und speicherintegrierte Rechnerarchitekturen sein, insbesondere in Anbetracht eines HMC. Dieses Problem wird jedoch aktuell nicht von Programmierparadigmen angegangen, sodass entweder die Anwendung oder aber das Laufzeitsystem diese Inkonsistenzen handhaben muss, um die maximale Bandbreite und dementsprechend die höchste Rechenleistung zu erzielen.

4.4 Trend

Es erscheint derzeit, als würden innovative Speicherarchitekturkonzepte zumeist in der Grafikdomäne als Erstes erscheinen oder zumindest adaptiert werden: Konzepte wie Vektoreinheiten, parallele Verarbeitungspipelines sowie das Multiplexen von Adressen mit Hilfe von *Scrambling*

als auch *Interleaving* über multiple Speicherchips und Speicherbänke hinweg wurden als Erstes im Bereich der Grafik eingesetzt [Rogers und Earnshaw, 1991]. Gerade das Multiplexen von Adressen ermöglicht, Speicherbänke nahezu zeitgleich parallel anzusteuern. Unter Umständen können physikalische Speicherseiten dabei innerhalb eines Speicherbankpuffers länger geöffnet bleiben (siehe *open / closed page policies*), wodurch Speicherzugriffe nicht nur über Parallelität, sondern auch durch einfaches Vorhalten von bereits gelesen oder geschriebenen Seiten beschleunigt werden können.

Zugleich wurde von Grafikexperten bereits früh versucht, die Effekte der *Bandwidth Wall* zu minimieren [Rogers und Earnshaw, 1991]. Als der Bildspeicher (engl.: *framebuffer*) bei Grafikkarten noch eine Limitierung zum Skalieren der Grafikverarbeitungsgeschwindigkeit darstellte und Adressmultiplexen nicht mehr ausreichte, wurde eine speicherintegrierte Rechnerarchitektur unter dem Namen *PIXEL-planes* entwickelt, um die Beschränkungen zu überwinden [Fuchs und Poulton, 1981]. Zur Zeit als Burger et al. auf die Limitierung der Speicherbandbreite und die daraus resultierenden Einschränkungen für zukünftige Mikroprozessoren hinwiesen [Kagi et al., 1996], re-evaluierten Kugler et al. bereits den Stand-der-Technik im Bereich der Speichertechnologie [Muller-Schloer et al., 1996, Kugler et al., 1996] und hoben hervor, dass Multiplexen von Adressen zu kostenintensiv sei und Speicher mit eingebetteter Logik, wie *FBRAM* [Deering et al., 1994b, Mitsubishi, 1996, Deering et al., 1994a] als auch *GRAMMY* [Knittel et al., 1996], die Zukunft darstellt [Knittel und Schilling, 1995]. Diese sollten explizit die Berechnungen beschleunigen, welche im Tiefenspeicher (engl.: *Z-buffer*) vorgehalten werden, da die Logik direkt in diesen eingebettet wurde. Hinzu kommt, dass Mitsubishi und Sun eine derartige speicherintegrierte Rechnerarchitektur mit dem Titel *3D-RAM*, auch bekannt als *FBRAM*, bereits in den 1990-er Jahren in kommerziellen Produkten vertrieb (z.B. Elite3D und Sun XVR-1000).

Im Bereich der dreidimensional gestapelten Speicher schritten ebenfalls Grafikkarten voran, bei welchem erstmals auf einer Fiji-Grafikkarte des Herstellers AMD (Radeon R9 Fury) vier Stapel aus *High-Bandwidth Memory* (HBM) mit je einem Gigabyte an Speicherkapazität aufgebracht wurden [AMD, 2015]. Dessen Nachfolger mit dem Titel HBM2 wurde ebenfalls im Bereich der Grafiksparte, d.h. in Verbindung mit Nvidias Pascal-GPU-Architektur (z.B. Nvidia Tesla P100) zuerst angeboten [Nvidia, 2016].

4.5 Zusammenfassung

Viele Studien beschäftigten sich mit der Einbettung von dedizierten Rechenelementen an Stellen, an welchen Daten in einem Rechensystem traversieren. Beispiele hierfür sind die Integration in Hauptspeicher, Cachehierarchie, Festplatten und Netzwerk. Schwerpunkt waren die planaren speicherintegrierten Verarbeitungsarchitekturen – sogenannte PIM –, welche Logik an verschiedenen Positionen direkt im DRAM-Speicher eingebettet hatten. Dabei konnte die technisch, technologische Machbarkeit illustriert und jeweils eine signifikante Rechenleistung aufgezeigt werden. Für letzteres war allerdings eine spezialisierte Verarbeitungsarchitektur dringend notwendig, konnte die durch den breiten Bus verfügbare, hohe Speicherbandbreite ansonsten nicht konsumiert werden. Hoch profitable Architekturen waren etwa funktionsgebunden (BPO), äußerst parallel, mit hoher Threadanzahl oder aber mit Vektorinstruktionen ausgestattet. Häufig konnte zusätzlich eine tiefe Cachehierarchie vermieden werden, wodurch eine Minimierung von energieineffizienten Datentransfers erzielt wurde.

Nicht gelöst war hingegen der aus der planaren Integration resultierende Zwang zur Wahl einer Prozesstechnologie, wodurch zwischen großer Speicherkapazität und geringer Rechengeschwindigkeit (DRAM-Technologie) oder umgekehrt kleiner Speicherkapazität und hoher Rechengeschwindigkeit (Logik-Technologie) entschieden werden musste. Gleichsam aus der tiefen Integration hervorgehend war die fehlende Möglichkeit unabhängig entweder Rechen- oder Speicherkapazität aufzurüsten. Gerade der Wahlzwang stellt angesichts moderner Fertigungsmethoden wie das dreidimensionale Stapeln für zukünftige PIM kein Problem mehr dar, können nun Logik und Speicher mit Hilfe von TSVs dicht integriert werden. Jedoch wird dies mit thermischer Belastung also Hitzeentwicklung erkauft. Die Akzeptanz von fehlender Aufrüstbarkeit erscheint hingegen im Gegensatz zur Vergangenheit gestiegen zu sein, werden doch einige hochspezialisierte Rechensysteme mit verlötetem Speicher wie etwa gestapeltem HBM vermarktet.

Gerne in Studien ausgelassen ist die Existenz einer MMU, welche Logik ausschließlich auf Cachezeilen, Seiten oder gar Segmente begrenzen könnte, oder die in einer zentralen Speicheransteuerung ausgeführte Adressverwürfelung über mehrere DRAM-Chips. Beide beeinflussen, neben der Integrationsstelle der Logik selbst, signifikant die Speichersicht, weshalb ein jeder Beschleuniger teils anders zu programmieren ist. Entsprechend wurde die PIM-Programmierung nicht eingehend untersucht und stellt noch immer ein großes Problem dar, obgleich heutzutage Programmiermodelle sowie APIs wie etwa OpenMP, CUDA und OpenCL existieren. Letztere fokussieren allerdings auf die Beschleunigerprogrammierung, nicht jedoch auf die vollständige Systemkomplexität, wodurch Effizienz und Rechenleistung minimiert werden. Ein möglicher Ausweg könnten neuere Konzepte wie das von der Firma AMD vorangetriebene HSA darstellen, da eine Abstraktion des einzelnen Rechenelements mit Hilfe der Aufteilung in Zwischenebenen, Speichermodell, Terminierungslogik und Laufzeit betrieben wird. In diese Umgebung passt gleichsam der HMC, welcher historische DDR-Signalisierung durch moderne Paketkommunikation abstrahiert.

Für die weitere Arbeit soll nun eine intendierte PIM-Architektur definiert werden, welche als Demonstrator der zu erstellenden PIM-Simulationsumgebung genutzt werden kann. Aufgrund der Vorteile der dreidimensionalen Integration, wird für die PIM-Simulationsumgebung ein Speicherstapel wie HMC in Betracht gezogen. Während die Speicherebenen unberührt bleiben sollen, soll die Logikebene mit der gewählten Verarbeitungsarchitektur ausgestattet werden. Die Integration derer findet vor dem internen Bus statt, wodurch eine vollständige Speichersicht auf den internen HMC ermöglicht wird, zugleich lokale sowie ferne *Vaults* adressiert werden können. Da programmierbar und mit hoher Threadanzahl auf Speicherlatenzvermeidung optimiert, bieten sich GPU-Architekturen zur GPGPU-Beschleunigung entsprechend als exzellenter Kandidat zur Integration an. Entsprechend soll eine adäquate GPU-Architektur im späteren Verlauf der Arbeit gewählt und für den Zweck der Demonstration der PIM-Simulationsumgebung integriert werden. Ein zentraler Prozessor kann angedacht werden, welcher ein vernetztes Geflecht aus mehreren PIM-Speicherstapeln als I/O-Prozessor ansteuert. Jedoch bedarf es an geeigneten Werkzeugen zur Modellierung und zur Simulation um solch PIM-Konfiguration zu erstellen. Dies soll im nachfolgenden Kapitel evaluiert werden.

Kapitel 5

Stand der Technik an Modellen

Im vorausgehenden Kapitel wurden verschiedenartigste PIM-Architekturen illustriert, welche jeweils versuchen *Memory Wall*, *Bandwidth Wall*, Kontaktstiftlimitierung und ineffiziente Datentransfers zu minimieren oder gar zu vermeiden. Zur Analyse von speicherintegrierter Beschleunigung sowie zur Entwurfsraumexploration von datennaher Rechenverarbeitung, werden Modelle benötigt, welche vollständige PIM-Architekturen oder zumindest Teile dessen abstrakt darstellen können. Da ein PIM ein komplexes System aus Speicher und Logik darstellt, muss eine vollständige PIM-Simulationsumgebung äußerst flexibel sein, beinhaltet dieses doch mindestens zwei eigenständige Simulationsmodelle. Demnach muss derartiges Werkzeug in Summe zur Modellierung ein ganzheitlich heterogenes Rechensystem abbilden können.

Zur Verkleinerung des Entwurfsraums wurde in Abschnitt 4.5 eine GPU-Architektur als Rechenbeschleuniger gewählt. Zeitgleich ist ein dreidimensionaler Speicherstapel vorgesehen, welcher mit Hilfe der GPU-Architektur zu einem PIM-Rechenbeschleuniger ausgebaut werden soll. Entsprechend widmet sich dieses Kapitel dem Stand der Technik an existierenden Simulationsmodellen und betrachtet hierzu einzelne Modellierungswerkzeuge für Speicher- und GPU-Architekturen, sowie Simulationsmodellen von heterogenen Systemen.

Die meisten Modelle stellen sogenannte Emulatoren sowie (funktionale) Simulatoren dar, welche teils als prozedurale oder teils als formale Beschreibungen vorliegen. Prozedurale Beschreibungen von Speicher- und GPU-Architekturen sind dabei häufig zeit- oder aktivitätsorientiert, weshalb zustands-, struktur- und datenorientieren Modelle kaum auszumachen sind. In Anbetracht aller stellt allerdings die primäre Ausrichtung die Modellqualität dar, wohingegen sekundär die Ausführungsgeschwindigkeit von Interesse ist. Entsprechend kommen Aspekte wie Eindeutigkeit, Verständlichkeit, Veränderbarkeit und Ausführungsgeschwindigkeit erst nachrangig zum Tragen. Klar ist, dass hohe Qualität eine präzise Modellierung bedingt und somit mit höherer Ausführungszeit einhergeht, wohingegen ungenaue Modellierung zu schnelleren Ergebnissen führt. Daher ist stets ein Kompromiss notwendig, um einerseits eine aussagekräftige Charakterisierung, andererseits eine schnell Simulation zu erzielen.

Da für eine generelle, jedoch auch für die nachfolgende Modellierung auf angemessene Modelle zurückgegriffen werden muss, werden Modelle von hohem Interesse für die nachfolgende Modellierung zumeist im Detail betrachtet.

5.1 Werkzeuge zur Speicherarchitekturmodellierung

Es existiert eine überschaubare Anzahl an Werkzeugen und Werkzeugsammlungen, welche als analytische Modelle, Emulatoren oder (funktionale) Simulatoren für Entwurf und Evaluation von Speichern vorliegen. Meist bieten diese eine Teilmenge der zu evaluierenden Parameter Speicherbandbreite, Latenz, Energieverbrauch, Leckströme, Fläche, Flüchtigkeit, Zuverlässigkeit und Temperatur. Obgleich die meisten dieser Werkzeugsammlungen zur Modellierung von flüchtigen als auch nichtflüchtigen Speicherarchitekturen bereits im Jahr 2002 existierten, zeigten Alakarhu et al. ein Mangel an zuverlässig vorhersagenden Softwarewerkzeugen auf, welche zukünftige neuartige Speicherarchitekturen abbilden können [Alakarhu und Niittylahti, 2002]. Dies ist bis zum aktuellen Zeitpunkt weiterhin der Fall, da auch die bereits etablierten Softwaremodelle meist neuartige Eigenschaften wie etwa das dreidimensionale Stapeln häufig nicht unterstützen, wodurch auch etablierte Werkzeuge nicht oder nur unzureichend für zukünftige Studien geeignet sind.

Aktuell können zwei Typen an Werkzeugsammlungen zur Speicherarchitekturmodellierung ausgemacht werden: Der erste Typ offeriert Modelle, welche Latenzen, Bandbreiten, Energieverbrauch und weitere Aspekte anhand von mathematischem Formelwerk berechnet, welche hiermit als rein analytische Modelle definiert werden. Der zweite Typ wiederum emuliert oder zumindest simuliert das Verhalten einer entsprechenden Speicherarchitektur. Diese werden meist mit Hilfe von sogenannten *Traces*, also Ablaufsequenzen von Speicherzugriffen, oder mit einem (integrierten) Prozessormodell betrieben. Da eine Vielzahl an Simulatoren, Emulatoren und Modellen existiert und für eine Studie die genutzte Variante von höchster Bedeutung ist, sollen die folgenden Abschnitte Einblicke in die Speicherarchitekturmodelle bieten.

5.1.1 Analytische Modelle

Das von HP labs entwickelte und frei verfügbare *CACTI* hat sich als De-facto-Standard und die erste Wahl auf Ebene der Schaltkreismodellierung von flüchtigen Speicherarchitekturen etabliert [Muralimanohar et al., 2007]. *CACTI* modelliert nahezu jeden Typ von flüchtigem Speicher, was nicht nur schnelle Vorhersagen ermöglicht, sondern auch die Analyse von unterschiedlichsten Aspekten, wie Speicherbandbreite sowie Speicherlatenz für neuartige Speicherarchitekturideen erlaubt. Seit Veröffentlichung der letzten offiziellen Version 6.5 Mitte des Jahres 2010 erscheint es allerdings, als wäre die Weiterentwicklung eingestellt worden. Trotzdem kann auch *CACTI* eine Modellierung von dreidimensional gestapelten Speicherforschung vornehmen. Hierzu wurde ein *CACTI-3DD* erstellt [Chen et al., 2012], welches in ein inoffizielles *CACTI 7.0-prerelease* integriert wurde. *CACTI-3DD* bietet im Bereich der dreidimensionalen Speicherstapelmodellierung das Modell eines Samsung High Bandwidth Memory (HBM), welches mit einer Standardkapazität von 8 GB ausgestattet ist, wobei eine Prozesstechnologie von 50 μm und eine Taktfrequenz von 667 MHz vorgegeben wurde. Im Gegensatz zu anderen analytischen Modellierungswerkzeugen, erlaubt *CACTI-3DD* eine flexible Anpassung der CMOS-Durchkontaktierungen, der TSVs.

Die Gruppe um Xie et al. veröffentlichte bereits viele nichtflüchtige Speichermodellierungswerkzeuge, welche den Entwurf von zukünftigen Speicherarchitekturen unterstützten. Der Startschuss kam mit *PCRAMsim* im Jahr 2007, welches sich – wie der Name bereits impliziert – auf *PCRAM* (*Phase-change random-access memory*) spezialisiert. Auf dem von *PCRAMsim* erbrachten Fundament aufbauend, jedoch offener gegenüber andersartigen Typen von nichtflüchti-

gen Speicherarchitekturen wie etwa SLC (*single-level cell*) NAND-Flash- oder ReRAM-Speicher, veröffentlichte dieselbe Gruppe das Modellierungswerkzeug *NVSim* im Jahr 2012 [Dong et al., 2012]. Aufgrund seiner freien Verfügbarkeit wurde *NVSim* als Basis für viele weitere Werkzeuge in der Forschungsgemeinde genutzt, da es einerseits Schaltkreismodellierung ermöglicht und andererseits gegen reale nichtflüchtige Speicherprototypen validiert wurde.

Bedingt dadurch, dass *CACTI-3DD* nie als Werkzeug frei verfügbar wurde und *CACTI* selbst ausschließlich Modellierung flüchtiger Speicher ermöglichte, entstand eine Lücke im Bereich der Werkzeuge zur Modellierung nichtflüchtiger Speicher, welche wiederum von *DESTINY* (*a 3D dEsign-Space exploraTIon tool for SRAM and eDRAM and Non-volatile memorY*) geschlossen wird [Mittal et al., 2015, Poremba et al., 2015a]. *DESTINY* ist eine Neuentwicklung des nationalen Laboratoriums in Oak Ridge und weiteren Kollaborateuren. Als Basis nutzt *DESTINY* das vorgestellte *NVSim* sowie Teile von *CACTI-3DD*, insbesondere hinsichtlich des Durchkontaktierungsmodells im Bereich des dreidimensionalen Stapelns. Somit stellt *DESTINY* Modelle von nichtflüchtigen und flüchtigen von dreidimensional gestapelten Speicherarchitekturen zur Verfügung. Da beide zugrundeliegenden Modellsammlungen signifikante Validierung gegen tatsächliche Hardware widerfahren haben, bildet das darauf aufbauende *DESTINY* ein vertrauenswürdiges, analytisches Speichermodellierungswerkzeug. Hinzu kommt, dass *DESTINY* auch als Ganzes gegen kommerzielle Prototypen validiert wurde, weshalb *DESTINY* als potentiell neuer Standard im Bereich der analytischen Speichermodellierung anzusehen ist; es ist daher zu erwarten, dass *DESTINY* den De-Facto-Standard *CACTI* mittelfristig ersetzt.

Mit dem speziellen Fokus auf Energieverbrauch im Bereich der Speicherentwurfsraumexploration wurde im Jahr 2011 *DRAMPower* vorgestellt. Im Detail ermöglicht *DRAMPower* die Modellierung von Stromverbrauch und Energiekonsum flüchtiger Speichern wie DDR2-, DDR3- und Wide-I/O-DRAM [Chandrasekar et al., 2011]. Wie zuvor bereits *DESTINY* wurde auch *DRAMPower* gegen tatsächliche Speicherhardware validiert.

5.1.2 Emulatoren und Simulatoren

Im Rahmen der Modellierung von Speichern können Werkzeuge für sowohl planare als auch dreidimensional gestapelte Architekturen ausgemacht werden. Diese werden im Folgenden im Detail betrachtet.

Planare DRAM-Modelle

In der Literatur findet sich *Rascas* als der erstgenannte DRAM-Simulator wieder, welcher eine Modellierung und Evaluation von verschiedenen Speicherbanktopologien erlaubt. Dabei ermöglicht *Rascas* eine Hintereinanderverschaltung von Speicherbänken oder gar ganze Baumstrukturen, welche wiederum mit entsprechenden Speicheransteuerungseinheiten verbunden sind [Alakarhu und Niittylahti, 2002]. Ähnlich zu *Rascas* fokussieren die Simulationswerkzeuge *DRAMsim* und *DRAMsim2* auf eine taktgenaue Simulation. Hierzu wurden die beiden letztgenannten gegen tatsächliche DRAM-Speichermakros (etwa DDR2 sowie DDR3) des Herstellers Micron mit Hilfe von Micron-eigenen Verilog-Makros verifiziert (Tabelle 5.1) [Rosenfeld et al., 2011, Wang et al., 2005]. Trotz intensiven Forschungsaufkommens im Bereich der Speicherarchitekturen und dessen Ansatz, die fein-granulare Taktgenauigkeit zu abstrahieren, um die Simulationsgeschwindigkeit und somit die Vorhersage von neuen Hardwareaspekten zu beschleunigen, sind *DRAMsim*

Simulationswerkzeuge	Open Source	Hardware Validierung	Funktionale Simulation	Taktgenaue Simulation	Planares DRAM-Modell	HMC-Modell	Multi-HMC-Vernetzung
Rascas	?			✓	✓		
DRAMsim & DRAMsim2	✓	✓		✓	✓		
DrSim	✓			✓	✓		
NVmain & NVmain2.0	✓	✓		✓	✓		
BOBSim	✓	✓		✓	✓		
Ramulator	✓	✓		✓	✓		
USIMM	✓			✓	✓		
(SST) CramSim	✓	?		✓	✓		
Active Memory Cube		?		✓		✓	?
CasHMC	✓			✓		✓	
HMC-Sim	✓		✓			✓	(✓)
(SST) VaultSimC	✓		✓			(✓)	
(gem5) Simple HMC model	✓	(✓)		✓		✓	
Vorgestellter Ansatz	✓	(✓)	✓	✓		✓	✓

Tab. 5.1: Funktionsübersicht von gängigen Simulatoren und Emulatoren für die Speichermodellierung.

als auch sein Nachfolger die Standardwerkzeuge im Bereich der Speicherarchitektur-Simulation, sofern die Genauigkeit der Vorhersage von höchstem Interesse ist.

Ein weiterer Vertreter der taktgenauen Simulation von Speicherarchitekturen ist *DrSim*, welcher im Gegensatz zu *DRAMsim* ein flexibles Werkzeug zur Adaption neuartiger Speichertopologien bereitstellt [Jeong et al., 2012]. Zum aktuellen Zeitpunkt wirkt es jedoch, als wäre keine Weiterentwicklung von *DrSim* angedacht. Dies ist auch der Fall für den Speicherarchitektursimulator *USIMM*, welcher eine taktgenaue Modellierung von unterschiedlichsten DRAMs ermöglicht [Chatterjee et al., 2012]. Im Gegensatz zu den zuvor genannten Werkzeugen bietet *USIMM* zugleich ein Prozessormodell an, welches *USIMMs* mit einer taktgenauen Simulation betreiben kann.

Die mit *DRAMsim* befasste Forschungsgruppe um Jacob etablierte den sogenannten, Hardware-validierten *BOBSim*-Simulator (engl.: *buffer-on-board*, BOB), welcher die Modellierung einer modernen Speicherhierarchie erlaubt [Cooper-Balis et al., 2012, Cooper-Balis, 2012]. Modern im Sinne von einer Cachehierarchie, bei welcher statt der klassisch bis zu drei Ebenen umfassenden internen Caches, ein weiterer großer Zwischenspeicher extern als L4-Cache auf der Platine eingebettet ist, was in Rechensystemen mit Hilfe von dedizierten Speicheransteuerungen wie z.B. dem IBM *Centaur*-Chip [Starke et al., 2015] oder Intel *Scalable Memory Buffer* (SMB) ermöglicht wird. Gerade dieser Mechanismus steht im Fokus des *BOBSim*-Simulators, welcher unterschiedlichste Topologien aus DRAM-Speicherbänken in Kombination mit externen Speicheransteuerungen modellieren kann. Khurshid et al. Erzeugten mittels *BOBSim* einen *Hybrid Memory Cube*, an welchem thermische Eigenschaften eruiert wurden [Khurshid und Lipasti, 2013]. Der in dieser Arbeit vorgestellte Ansatz geht über den von Khurshid et al. signifikant hinaus, indem nicht nur die Speicherschichten, sondern auch die Logikschicht des HMCs modelliert wird. Dies erlaubt eine detaillierte und wirklichkeitsgetreue Simulation.

In den Jahren 2012 und 2015 wurden *NVmain* und *NVmain2.0* vorgestellt, um die damals monolithischen Simulationsansätze und deren Defizite zu überwinden. Diese Schwächen waren insbesondere bei der Adaption von neu aufkommenden flüchtigen und nichtflüchtigen Speichern begründet [Poremba et al., 2015b]. *NVmain* führte hierzu das Konzept einer Aufteilung in Komponenten ein und ahmt somit tatsächliche Speicher nach, sodass die Speicheransteuerung, Speicherverbindungen als auch DRAM-Speicherbänke in jeweils simulierte Softwaremodulklassen gekapselt wurden. Dadurch wurde die Integration von neuartigen Typen von Speichern und entsprechend die Simulation von Speicherarchitekturen vereinfacht. Derselbe Gedanke wurde im Jahr 2016 mit dem Simulator *Ramulator* etabliert, welcher eine abstrakte, moderne C++-basierte Simulationsumgebung bietet und zugleich mit einem umfassenden Angebot von gängigen sowie exotischen flüchtigen Speicherarchitekturen aufwarten kann [Kim et al., 2016]. Wohingegen althergebrachte Simulationswerkzeuge nicht explizit mit Fokus auf Simulationsgeschwindigkeit entwickelt wurden, wurde bei *Ramulator* eine hierarchische Baumstruktur eingebettet. Dies verhilft *Ramulator* dazu, nur die Komponenten des Simulators je Takt zu aktualisieren, welche tatsächlich einen Bedarf haben. Entsprechend übertrifft *Ramulator* bei der Simulationsgeschwindigkeit althergebrachte Ansätze hinsichtlich der Ausführungsgeschwindigkeit. *Ramulator* selbst jedoch folgte seinem Vorgänger *Ramulator#*, welcher nahezu die gleiche Funktionalität bietet, jedoch zuvor in C# statt C++ entwickelt wurde [Chang und Kim, 2016].

Die "strukturierte Simulationswerkzeugsammlung" SST (engl.: *The Structural Simulation Toolkit*, SST) strebt an, die Standardplattform für die Forschung an Rechnerarchitekturen zu

werden, indem es ein ganzheitliches, frei verfügbares sowie hoch parallelisiertes Simulationswerkzeug darbietet [Rodrigues et al., 2011]. Obgleich der Fokus nicht explizit auf Speicherarchitekturen gelegt wurde, wurden bereits verschiedene Speichersimulatoren als auch analytische Modelle explizit für SST entworfen, wodurch ein breiteres Spektrum zur Charakterisierung von Rechnerarchitekturen betrachtet werden kann. Hierzu zählt IBMs *CramSim* (*ControlleR And Memory SIMulator*), welcher partiell den bereits vorgestellten Simulator *USIMM* nutzt, um moderne DDR3- und DDR4-Speicher zu simulieren.

Modellierung von dreidimensionalen Speicherstapeln

Erstmals wird eine dreidimensionale Simulation eines Speicherstapels im Zusammenhang mit dem *Active Memory Cube* genannt; diese ist allerdings nicht frei verfügbar (Unterabschnitt 4.2.3) [Nair et al., 2015]. Ebenfalls nicht frei verfügbar ist das von der Gruppe um Jacob et al. entwickelte HMC-Simulationsmodell für Micron [Rosenfeld, 2014], welches bereits Softwareschnittstellen innerhalb des SST-Simulationswerkzeugsammlung erhalten hat [Rodrigues et al., 2011]. SST selbst beinhaltet bereits mit *VaultSimC* ein äußerst vereinfachtes Speicherstapelmodell.

Im Zusammenhang mit dem Vielkernprozessor *GoblinCore64*, welcher auf der RISC-V-Architektur der UC Berkeley basiert, wurde ein HMC-Simulationsmodell mit dem Titel *HMC-Sim* erstellt [Leidel und Chen, 2014, Leidel und Chen, 2016]. *HMC-Sim* selbst unterstützt nur eine funktionale Simulation, bei welcher Lade- und Speicheroperationen nicht auf elementare Datenpakete (engl.: *flow control unit*, FLIT) heruntergebrochen werden. Zugleich konzentriert sich *HMC-Sim* auf die Modellierung eines einzigen HMC, wobei ein Multi-HMC-Szenario für die Zukunft vorgesehen ist. Seinen Vorteil kann *HMC-Sim* hinsichtlich neuartiger In-Speicherinstruktionen ausspielen, welche vom Nutzer jeweils individuell als Bibliothek kompiliert werden, um sie daraufhin dynamisch zur Laufzeit in das HMC-Simulationsmodell einzuladen. In-Speicherinstruktionen selbst stellen tatsächlich Recheninstruktionen dar, welche durch die Verarbeitungspipeline eines Rechenkerns als Lade- oder Speicheroperation an einen Speicher delegiert werden, innerhalb dessen jedoch als Arithmetik ausgeführt wird.

Wohingegen die bereits angesprochenen Softwaresimulatoren keine Validierung gegen Hardware erfahren hat, wurde von Azarkhish et al. ein auf dem AXI-Protokoll basierender Hardware-Switch entwickelt, welcher in der untersten Ebene d.h. in der Logikschicht eines selbst erstellten, nicht standardisierten HMCs Verwendung findet [Azarkhish et al., 2015]. Zugleich wurde ein nicht spezifikationsgetreuer Speicherstapel aus 4 GBit großen DRAM-Speichern mit jeweils sechzehn unabhängig voneinander arbeitenden Zonen (*Vaults*) erstellt, welche über die geschichteten Speicherbankebenen mit der Logikebene verbunden wurden. Der so entstandene Spezial-HMC wurde später als Softwaresimulator mit dem Titel *simple HMC model* in den De-facto-Standardrechenarchitektursimulator *gem5* integriert [Binkert et al., 2011]. Hinzu kam, dass Azarkish et al. eine sogenannte *SMCSim*-Umgebung errichtete, welche vollständige Systemanalyse von speichernahen Verarbeitungseinheiten ermöglichte, indem wiederum das *simple HMC model* mit einem ARM-Rechenkern kombiniert wurde [Azarkhish et al., 2016]. Dieser dient entsprechend als speicherintegrierter Beschleuniger.

Das jüngste taktgenaue HMC-Simulationsmodell wurde von Jeon et al. unter dem Titel *CasHMC* vorgestellt, welcher ähnliche Entwurfsziele wie diese vorliegende Studie verfolgt [Jeon und Chung, 2016]. *CasHMC* spezialisiert sich jedoch auf die Evaluierung von Zuverlässigkeit und

Validierung. Entsprechend wurde die vollständige Schnittstellenlogik, welche die Verwürfelung und "Entwürfelung" (engl.: *scrambling & descrambling*) von Daten auf dem Bus mit einschließt, implementiert. Somit kann dieser Verbindungsabbrüche zwischen den Schnittstellen und die Effekte aus Fehlerinjektionen eruieren.

Wie zu erkennen ist, existiert nur eine überaus beschränkte Anzahl an Simulationsmodellen von dreidimensionalen Speicherarchitekturen, welche auch nur – zum Teil sehr – bedingt taktgenau sind. Auch sind Simulatoren im Bereich von Multi-HMC-Umgebung nicht gegeben. Bedingt durch die Beschränkung auf einen Speicherhersteller im HMC-Bereich (Micron) und nur wenigen hinsichtlich HBM, gibt es keine aussagekräftige Spezifikationen. Dadurch mangelt es allen hier veranschaulichten Werkzeugen zur Modellierung von dreidimensionalen Speicherstapeln an einer Validierung gegen reale Hardware. Als einziger Ausweg erscheint die Modellierung von dreidimensionalen Stapeln unter Zuhilfenahme von bereits validierten, planaren DRAM-Speichermodellen und dreidimensionalen Charakteristik-Annahmen zum Aufbau eines Speicherstapels.

5.2 Simulationswerkzeuge zur Modellierung von GPGPU-Rechenbeschleunigern

Frühe Forschung im Bereich der speichernahen Verarbeitung zeigte den signifikanten Mehrwert an maßgeschneiderten Rechenarchitekturen bei der Integration in Speicherarchitekturen auf (Abschnitt 4.5). Unter mehreren wurden insbesondere GPU-Architekturen als vorteilhaft identifiziert, jedoch bedarf es hinreichend akkurate Simulationsmodelle um diese für eine zukünftige dreidimensionale PIM-Studie zu nutzen. Neben dem Ziel einer hohen Modellqualität sind seitens eines Simulationsmodells auch Ausführungsgeschwindigkeit sowie die Rückführung von neuen Erkenntnissen von belang. Wohingegen die Geschwindigkeit mit darauf optimierten, unter Umständen rein funktionalen, Simulationsmodellen entgegen getreten werden kann, stellt die Rückführung aufgrund der Modellierung von proprietären GPU-Architekturen eine Herausforderung dar. Daher werden unter anderem GPU-Architekturen mit vorhandener RTL-Implementierung in Erwägung gezogen, welche in eine holistischen PIM-Entwurfsraumexploration flexibel eingebracht und anhand der jeweiligen Bedürfnisse angepasst werden können. Beispielhaft genannt seien zwei essentielle, jedoch gegensätzliche Entwurfsaspekte bei der Integration von GPU-Kernen in eine PIM-Architektur: Aus der Toleranz hoher Speicherlatenzen resultiert die Anforderung nach multiplen Recheneinheiten, welche ihrerseits mehrere Hardware-Threads zur Verfügung stellen; dies steht jedoch diametral zur für Logikelemente zur Verfügung stehenden Chip-Fläche. Hieraus ergeben sich die widerstreitenden Anforderungen nach maximaler Anzahl von Kernen im Gegensatz zu äußerst beschränkter Fläche. Dies ist ein zentraler Punkt der holistischen PIM-Entwurfsraumexploration. Hierzu benötigt wird insbesondere ein flexibler Rechenkern, der die genannten Forderungen hinsichtlich Multicore- und Multithreading-Eigenschaften erfüllt und der sich entsprechend der Entwurfsraumexploration anpassen lässt.

Das nachfolgende Unterkapitel widmet sich daher dem Stand der Technik hinsichtlich existierender Modelle von GPU-Rechenbeschleunigern. Hierzu werden einerseits die reinen Simulationsmodelle entsprechend ohne RTL-Implementierung aufgezeigt, sowie Modelle bei welchen auch Hardware mit Hilfe von frei verfügbarer RTL-Implementierung synthetisiert werden kann. Dabei soll geklärt werden, welches Modell eines GPGPU-Rechenbeschleunigers sich für die nach-

folgende PIM-Studie qualifiziert.

5.2.1 Softwaresimulatoren zur Modellierung von GPGPU-Rechenbeschleunigern

Titel	Verfügbares Modell	ISA
GPGPU-Sim	Taktgenauer Simulator	Nv. 8600GTS
SIMT-GPU	Taktgenauer Simulator	Nv. 9600GT
Barra (UNISIM)	Paralleler funktionaler Simulator	Nv. Tesla
Gputejas	Paralleler Ablaufsequenz-gesteuerter Simulator	Nv. Tesla
Ocelot	JIT-kompilierter Emulator	CUDA

Tab. 5.2: In der Literatur verfügbare Modelle an GPGPU-Rechenbeschleunigern ohne RTL-Implementierung.

Aktuell existiert nur eine kleine Auswahl an frei verfügbaren Simulationswerkzeugen, welche die Fähigkeit besitzen, Grafikkarten oder gar Beschleuniger wie die frei programmierbare Grafikprozessoren (GPGPU) hardwaregetreu zu modellieren (Tabelle 5.2). *GPGPU-Sim* ist dabei der bekannteste GPU-Simulator, welcher von Forschern für verschiedenste Zwecke im Bereich der architektonischen Entwurfsraumexploration genutzt wird [Bakhoda et al., 2009]. Hierzu bietet *GPGPU-Sim* eine funktionale als auch eine taktgenaue Ausführung an, sodass eine schnelle Vorhersage oder eine detailreiche Analyse von Rechenleistung von neuartigen GPGPU-Rechenbeschleunigern erstellt werden kann. Obgleich eine Vielzahl weiterer Aspekte wie z.B. Energieverbrauch, Zuverlässigkeit, Speicherbandbreiten und Speicherlatenzen betrachtet werden können, fehlt eine RTL-Implementierung, wodurch keine validierte, sondern ausschließlich eine annäherungsweise Hardwarecharakterisierung möglich ist. *GPGPU-Sim* wurde zudem bereits parallelisiert, wodurch eine erhöhte Simulationsgeschwindigkeit erzielt wurde [Lee und Ro, 2013].

Der Grafikkartensimulator *SIMT-GPU* wurde von Wang et al. nicht explizit für die Zwecke der GPGPU-Allzweckberechnung, sondern primär auf eine detailreiche Beschleunigung von grafischen Anwendungen wie OpenGL ES entwickelt [Wang et al., 2012]. Dabei implementiert *SIMT-GPU* alle klassischen Architekturdetails einer Grafikkarte wie z.B. das Laden und Ausführen von Instruktionen, als auch das sogenannte *Warp Scheduling* oder gar die Textur-Pipelines. Ein modulares als auch parallel, funktionales Simulationsmodell einer GPU-Architektur wurde von Collange et al. unter dem Namen *Barra* vorgestellt [Collange et al., 2010], welcher auf dem nahmhaften *UNISIM* Modellierungswerkzeug aufbaut [CEA List institute, 2011]. *Barra* modelliert dabei eine Nvidia-Grafikkarte und kann aufgrund von binärer Kompatibilität kompilierten CUDA-Quelltext nativ ausführen. Trotz dem vielversprechenden Ansatz seitens des *Barra*-Werkzeuges auf das *UNISIM* und entsprechend auf die Sprachen SystemC sowie TLM aufzusetzen, existiert keine RTL-Implementierung des Simulationsmodells, wodurch erzielte Ergebnisse nicht validiert werden können.

Im Gegensatz zu den bereits genannten Ansätzen, bei welchen eine ganzheitliche Modellierung einer Grafikkarte verfolgt wird, erstellten Malhotra et al. einen ablaufsequenzgesteuerten (engl.: *trace-driven*) funktionalen Emulator, welcher durch hochgradige Parallelisierung eine signifikant beschleunigte Simulationsgeschwindigkeit ermöglicht [Malhotra et al., 2014]. Ein wei-

teres orthogonales Feld wurde mit dem Emulator *Ocelot* erbracht, welcher CUDA-Quelltext zur Laufzeit per JIT-Kompilierung auf die Zielarchitektur übersetzt [Diamos et al., 2010].

5.2.2 GPGPU-Rechenbeschleuniger mit RTL-Implementierung

Bereits in der Literatur, jedoch auch im freien Umfeld, sind RTL-Implementierungen von Grafikkartenbeschleuniger noch rarer als die Softwaresimulatoren vertreten. Prinzipiell können fünf

Titel	**Verfügbares Modell**	**ISA**	**Aktiv**
OpenShader	Emulator, Verilog-RTL	Neuentwicklung	-
Theia	Verilog-RTL	Neuentwicklung	-
MIAOW	Verilog-RTL, Simulator	AMD-*Southern-Island*	✓
NYUZI (ehemals: NYAMI)	SystemVerilog-RTL	Neuentwicklung	✓
FlexGrip	VHDL-RTL	CUDA-kompatibel	-

Tab. 5.3: Literarisch verfügbare Modelle von GPGPU-Rechenbeschleunigern, welche in einer RTL Sprache implementiert sind.

solcher RTL-Implementierungen ausgemacht werden, wobei bei zumindest zweien bereits keine weitere aktive Entwicklung stattfindet (Tabelle 5.3). Eine davon ist die voll funktionstüchtige Implementierung einer GPU namens *OpenShader*, welche in Verilog geschrieben und gleichzeitig als Emulator verfügbar ist [Miller, 2015]. Ebenso verfügbar ist eine RTL-Implementierung namens *Theia*, welche einen in Verilog geschriebenen, auf Strahlenverfolgung (engl.: *raytracing*) spezialisierten Grafikkartenprozessor anbietet. Zur Beschleunigung der graphischen Bildsynthese besitzt die Architektur dedizierte SIMD-ALUs [Valverde, 2015]. Andryc et al. illustrierte eine RTL-Implementierung eines CUDA-fähigen, jedoch auf Ganzzahlarithmetik limitierten GPGPU-Rechenbeschleuniger [Andryc et al., 2013]. Zur Demonstration wurde die GPU-Architektur auf einem FPGA mit einem Xilinxs MicroBlaze-Mikroprozessor angetrieben und validiert, jedoch ist der Quellcode der RTL-Implementierung nicht frei verfügbar.

Die vielversprechendste RTL-Implementierung stellt *MIAOW* dar, welche von Balasubramanian et al. an der Universität Wisconsin-Madison entworfen wurde [Balasubramanian et al., 2015]. Der Rechenkern der Grafikkarte baut dabei auf den Typ AMD-*Southern-Island* auf, wodurch diese binärkompatibel ist und somit den kompletten AMD-Softwarestack ausführen kann. Zugleich bietet die Gruppe eine evolutionäre, dreistufige Simulation an. Bei dieser wird je nach Forschungsaspekt entweder die komplette RTL-Implementierung, Teile der RTL-Implementierung mit Softwaresimulationskomponenten oder ausschließlich Softwaresimulation genutzt. Da *MIAOW* aus einem Forschungsprojekt hervorgegangen ist, soll explizit *MIAOW* keine Konkurrenz zu etablieren Hardwareherstellern darstellen, jedoch bietet *MIAOW* bereits Rechenleistung auf dem Niveau von aktuellen, jedoch langsamen proprietären GPGPU-Rechenbeschleunigern. Einen Aspekt dessen stellt der grafische Ausgang dar, welcher in der Theorie durch einen spezialisierten IP-Block realisiert werden kann, der jedoch nicht zur Verfügung steht.

Im Kontrast zu *MIAOW* steht die RTL-Implementierung *NYUZI* (ehemals: *NYAMI*), welche ebenfalls einen Architekturentwurf jedoch mit neu entworfener ISA bereitstellt [Bush et al., 2015]. Diese wurde stark von dem MIC-Vielkernprozessoren vom Typ Intel Xeon Phi sowie dem GPGPU-Paradigma beeinflusst. Bedingt durch die GPU-Ausrichtung besitzt *NYUZI* eine

	NYUZI-Rechenkern	AMC-VLIW-*Lane*
Eigenschaften	4 Hardware-Threads 16 FX/FP$_{sp}$ Einheiten 1 LD/ST Einheit	9 Funktionseinheiten 4 FX/FP$_{dp}$ / 8 FP$_{sp}$ Einheiten 4 LD/ST Einheiten
# an $\frac{FLOP_{sp}}{Takt}$	16	16 (mit *multiply-add*)
Frequenz [MHz]	570	1250
Technologie [nm]	45	45
Registerfile [Byte]	$10880_{32\,\text{Bit}}$ *(19584* $_{64\,\text{Bit}}$*)*	$17920_{64\,\text{Bit}}$
Instruktionszwischen-speicher [Byte]	16384 (L1i)	$\leq$ 18432 (*LIB*, 512 Instruktionen)
Datenzwischen-speicher [Byte]	16384 (L1d)	-

Tab. 5.4: Gegenüberstellung des Ressourcenbedarfs eines *NYUZI*-GPU-Rechenkerns zu einer AMC-VLIW-*Lane*.

Vielzahl an Rechenkernen und Hardware-Threads und ist daher unempfindlich gegenüber hohen Speicherlatenzen. Da der Fokus beim Entwurf auf hochgradiger Rechengeschwindigkeit in Kombination mit kleinem Flächenanspruch war, kann jedoch die *NYUZI*-Architektur selbst in kleinen, günstig verfügbaren FPGAs genutzt werden. Tabelle 5.4 zeigt hierzu einen Vergleich zwischen einem *NYUZI*-Rechenkern gegenüber eine auf Fließkommaberechnungen sowie Energieeffizienz getrimmte VLIW-Recheneinheit eines AMC-Speicherstapel auf (Unterabschnitt 4.2.3). Im Vergleich dazu ist *MIAOWs* RTL-Implementierung äußerst groß, wodurch selbst ein Xilinx Virtex-7 FPGA ausschließlich einen einzigen Rechenkern halten kann [Balasubramanian et al., 2015].

5.3 Modellierungswerkzeuge für heterogene Systemarchitekturen

Titel	**Prozessormodell**	**Beschleunigermodell**
gem5-gpu HSAemu	gem5 PQEMU	GPGPU-SIM - Teile aus multi2sim GPU - eigenes GPU-Modell
MacSim	Ablaufsequenzgesteuerter HSA-Simulator	
FusionSim	PTLsim + MARSSx86	Teile aus GPGPU-Sim
Jang et al.	eigener Simulator, ähnlich zu AMD Llano APU	

Tab. 5.5: Modellierungswerkzeuge für heterogene Systemarchitekturen.

Soll eine Modellierung eines heterogenen Simulationsmodells, existierend aus Hauptprozessor und Beschleuniger, wie z.B. eine Grafikkarte, getätigt werden, so existiert in der Literatur nur eine kleine Anzahl an Werkzeugen. Hintergrund ist, dass es nur wenige GPGPU-Rechenbeschleunigermodelle gibt (Unterabschnitt 5.2.1), wodurch noch weniger heterogene Simulationsmodelle verfügbar sind. Eine Auswahl der gängigsten wird in Tabelle 5.5 aufgezeigt.

Eines dieser Simulationsmodelle stellt *HSAemu* dar, welches aus dem parallel simulieren-

den Prozessormodell *PQEMU* und der taktgenauen Grafikeinheit *multisim* erstellt wurde. Zur schnelleren Simulation existiert ein selbst erstelltes funktionales Grafikkartensimulationsmodell [Ding et al., 2014]. *HSAemu* zielt auf die Entwicklung von schnellen heterogenen Algorithmen, welche die Fähigkeiten dieser Plattform auszuschöpfen vermag. *multi2sim* selbst stellt ebenfalls ein heterogenes Modellierungswerkzeug dar, welches einen schnellen funktionalen als auch einen detailgenauen Architektursimulator in Form eines x86-Prozessors und einer GPU vom Typ AMD-*Evergreen* bietet [Ubal et al., 2012]. Hauptaugenmerk von *multi2sim* ist die taktgenaue Modellierung der einzelnen heterogenen Komponenten.

GPGPU-Sim stellt den wohl bekanntesten Simulator für Grafikkarten dar; im Bereich von Prozessoren ist dies hingegen *gem5* [Binkert et al., 2011]. Beide Simulationsmodelle kombiniert erzeugte einen mächtigen heterogenen CPU-GPU Simulator, welcher den Titel *gem5-gpu* trägt [Power et al., 2014]. Es erscheint als würde *gem5-gpu* nahezu jeglichen Forschungsaspekt für holistische Architekturentwurfsraumexploration erfüllen, baut er doch auf den bekanntesten und stark vorangeschrittenen Simulatoren für die jeweilige Domäne auf. Von Kim et al. stammt hingegen der Simulator *MacSim*, welcher Ablaufsequenzen zur Simulation von heterogenen Systemarchitekturen verwendet; dies steht im Gegensatz zu *gem5-gpu* welcher rein auf eine tatsächliche Ausführung (engl.: *execution-driven*) setzt [Kim et al., 2012].

Zakharenko et al. untersuchten Rodinia-Benchmarks und entwickelten dazu den SoC-Simulator *FusionSim* [Zakharenko et al., 2013]. *FusionSim* baut gleichermaßen auf *GPGPU-Sim* für die Grafikkarte, *PTLsim* für den Prozessor und hinsichtlich der Cachehierarchie auf *MARSSx86*. Ähnlich zu Zakharenko et al. generierten Jang et al. einen heterogenen SoC-Simulator, welcher den Architekturansatz einer *Accelerated Processing Unit* (APU) der Firma AMD besitzt [Jang et al., 2015]. Hauptaspekt ihres Simulators ist die Evaluierung von Energiekonsum, da ein hochparalleler Prozess mit Hilfe AMDs-HSA-Architektur je nach Energiebudget eines SoC entweder auf Prozessor oder Grafikkarte terminiert werden kann.

Obgleich die vorgestellten Modellierungswerkzeuge zur Simulation von heterogenen Architekturen einen holistischen Entwurf erlauben, bieten diese ausschließlich ein Softwaremodell und keine RTL-Implementierung. Entsprechend ermöglichen diese die Modellierung sowie eine Entwurfsraumexploration, jedoch fällt es schwierig, neue Hardwaredetails die in der Simulation als vielversprechend identifiziert wurden in tatsächlicher Hardware zu validieren.

5.4 Zusammenfassung

Im vorliegenden Kapitel wurden unterschiedliche Werkzeuge und Simulatoren zur Modellierung von Speicherarchitekturen, GPGPU-Rechenbeschleunigern als auch heterogenen Rechnerarchitekturen vorgestellt und anhand der Verwertbarkeit innerhalb einer PIM-Studie betrachtet. Hintergrund ist, dass ein PIM eine komplexe heterogene Architektur darstellt, weshalb bei einer PIM-Simulation Modelle für eine Speicher- sowie eine Beschleunigerarchitektur benötigt werden. Wohingegen mehrere Speicherarchitekturmodelle existieren, stellen Modelle für GPU-Architekturen eine Nische dar, sodass nur wenige verfügbar sind. Da heterogene Rechenarchitekturwerkzeuge auf diesen wenig verfügbaren GPU-Architekturmodellen aufbauen, teilen diese ein ähnliches Schicksal, sodass gerade einmal vier existieren.

Dreidimensionales Stapeln von Speicherbänken, als das kritische Hardwaredetail ei-

ner dreidimensionalen PIM-Architektur, wird ausschließlich von fünf aus insgesamt 13 Speichersimulationsmodellen unterstützt. Diese sind jedoch bis auf eine Ausnahme nicht gegen Hardware validiert, wobei selbst die Ausnahme mit dem Titel *(gem5) Simple HMC model* nicht gegen einen standardkonformen HMC-Speicherstapel evaluiert wurde. Während *(gem5) Simple HMC Model*, *CasHMC* und *(SST) VaultSimC* noch auf elementarer Datenpaketebene (FLIT) modellieren, bietet keines der verfügbaren eine Multi-HMC-Umgebung zur Simulation von vernetzten HMCs. Entsprechend ist eine aussagekräftige Simulation von dreidimensionalen Speichern nur äußerst erschwert möglich. Einen Ausweg bietet das Simulationsmodell *BOBSim* an, indem dieses Speicherbänke von der Speicheransteuerungen trennt, um eigentlich primär neuartige, planare Off-Chip-Speicherarchitekturen darzustellen. Entsprechend kann *BOBSim* mit Anpassungen einen HMC-Speicher modellieren, weshalb Khurshid et al. dies bereits in Auszügen (d.h. im Detail ohne Switch) getätigt hatten. Wird von dreidimensionaler Speicherstapelung abgesehen, so wurde ein weiter wichtiger Aspekt durch den Speichersimulator *Ramulator* aufgezeigt. Dieser verwendet zur Beschleunigung der Simulationsausführungsgeschwindigkeit eine interne Baumstruktur, um pro Takt nur die Teile eines Speichers zu simulieren, welche einen tatsächlichen Bedarf haben.

Wie beschrieben stellen Modelle für GPU-Architekturen eine Nische dar, sodass gerade einmal fünf Softwaresimulatoren und fünf Modelle mit RTL-Implementierung existieren. In Anbetracht einer PIM-Studie, bei welcher ein GPGPU-Rechenbeschleuniger in einen HMC-Speicherstapel integriert werden soll, erscheint alleinig der neuartige GPGPU-Rechenbeschleuniger *NYUZI* von Interesse; primär aufgrund der Qualität der Beschreibung, welche durch RTL einerseits in einen Emulator überführt und ausgeführt sowie andererseits in einen ASIC synthetisiert werden kann. Sekundär, bedingt durch dessen Flexibilität, Verständlichkeit und Veränderbarkeit, wodurch unterschiedlichste PIM-Szenarien angedacht werden können. Zugleich zielt *NYUZI* im Gegensatz zu anderen RTL-Implementierungen auf den Bereich der eingebetteten Systeme, weshalb dieser nur einen geringen Ressourcenbedarf besitzt und sich somit für die Integration in eine HMC-Logikebene anbietet. Auch ist *NYUZI* keine GPU-Architektur im klassischen Sinne, sondern wurde von Vielkernprozessoren wie Intels MIC Larrabee beeinflusst, wodurch er für die Rechenbeschleunigung vielversprechend erscheint.

Die Nische von GPU-Simulationsmodellen teilend existieren gerade einmal vier Werkzeuge zur Modellierung von heterogenen Rechensystemen. Hauptproblem derer ist die Fokussierung auf Prozessor und Beschleuniger, nicht jedoch auf Beschleuniger und Speicher. Ohnehin ist durch insgesamt hohe Abstraktion und teils Extraktion aus gängigen Beschleunigermodellen fraglich, ob gewonnene Erkenntnisse tatsächlich in reale Hardware überführt werden können. Entsprechend stellen derartige heterogene Rechnerarchitekturmodelle für einen PIM-Entwurf keine geeignete Option dar.

Kapitel 6

Modellierung

Zur Modellierung eines ganzheitlichen Modellierungswerkzeugs für dreidimensionale PIM-Architekturen bedarf es gleichermaßen Simulationsmodelle der Speicherstapelung sowie der als Beschleuniger genutzten Recheneinheiten. Hinsichtlich der Analyse großer Problemstellungen ist hierbei eine hohe Simulationsgeschwindigkeit unabdingbar; diese darf jedoch nicht zu Lasten der Simulationsgenauigkeit gehen. Weiterhin ist eine maximale Flexibilität des Modellierungswerkzeugs erforderlich, um dem Anspruch an ein holistisches Modellierungswerkzeug zur PIM-Entwurfsraumexploration gerecht zu werden.

Um diese nur schwer in Einklang zu bringenden Ansprüche zu erfüllen, bedient sich das entwickelte Modellierungswerkzeug folgender Teilmodelle: Ein auf der HMC-Spezifikation in der Version 2.1 basierendes HMC-Modell liefert das Simulationsmodell für den dreidimensionalen Speicherstapel. Das Modell des Rechenbeschleunigers ist aus der *NYUZI*-Instruktionssatzarchitektur abgeleitet. Beide Modelle wurden explizit für diese Arbeit zur Integration in das PIM-Explorationswerkzeug entwickelt, da die Recherche hinsichtlich des Stands der Technik keine für diese Arbeiten verwendbaren oder weiterentwickelbaren Modelle ergab.

Zur Wahrung der Genauigkeit und um eine aussagekräftige Charakterisierung zu ermöglichen, wird seitens des HMC-Speichermodells für die Simulation der *Vaults* und damit den Speicherpartitionen sowie -bänken auf den Simulator *BOBSim* zurückgegriffen. Dieser ist gegen real existierende Hardware evaluiert und somit in der Lage, belastbare realitätsgetreue Simulationsaussagen zu tätigen. Entsprechend wurde für das GPU-Beschleunigermodell auf die in SystemVerilog vorliegende, frei verfügbare RTL-Implementierung zurückgegriffen, um das entwickelte Simulationsmodell zu validieren.

Das resultierende, flexibel parametrisierbare Modellierungswerkzeug wird in den nachfolgenden Unterkapiteln detailliert dargestellt. Da die HMC-Spezifikation eher grundsätzlicher Natur, entsprechend keine Angaben hinsichtlich maximal anzusetzender Ressourcen macht, und die Integration eines Beschleunigers nicht thematisiert ist, wird als vergleichendes Fallbeispiel der Active Memory Cube herangezogen; die Ressourcen des AMC sind aus der Literatur hinreichend bekannt, wie in Unterabschnitt 4.2.3 beschrieben. Zusätzlich wird eine mögliche Integration der einzelnen Teilmodelle sowie des Modellierungswerkzeugs in etablierte virtuelle Plattformen wie SST und *SoCRocket* aufgezeigt.

6.1 *Hybrid Memory Cube* (HMC)

Frei verfügbare HMC-Spezifikationen sind äußerst ungenau bis schwammig formuliert, wodurch eine präzise Modellierung schwierig ist. Hiervon insbesondere betroffen ist die Logikebene sowie die interne Vernetzung zwischen Quadranten, Aufbau der Quadranten, der *Vaults*, die Latenzen und Frequenz. Entsprechend wurde auf Basis der vorhandenen Angaben ein HMC-Simulationsmodell implementiert (Tabelle 3.4), welches flexibel genug ist, um genauere Details in der Zukunft einzuarbeiten.

Grundsätzlich wurde das HMC-Simulationsmodell auf der elementaren Datenpaketebene, entsprechend FLIT-Ebene, implementiert. Hierzu wurde ein dedizierter Routing-Algorithmus notwendig und entwickelt, welcher innerhalb des HMCs FLITs führen kann. Zeitgleich konnte im Gegensatz zu allen anderen HMC-Simulationsmodelle eine Multi-HMC-Umgebung geschaffen werden, bei der eine Traversierung von FLITs innerhalb des vollständigen, somit auch externen, Netzwerks getätigt werden kann. Da das Aufsetzen und Anpassen von Parametern für Netzwerk, Netzwerkswitch sowie Schnittstellencharakteristik schwierig ist, wurde ein vereinfachtes Einspielen von derartigen Konfigurationen mit Hilfe einer Graphbeschreibungssprache ermöglicht.

Auf Grundlage der ungenauen Spezifikation im Bereich des Quadranten wurde eine eigene Arbitrierungslogik implementiert, welche je nach eingehender Anfrage auf einem Eingangsanschluss den passenden Ausgangsanschluss auswählt und diesen auf Kollision überprüft. Sofern zutreffend wird die Anfrage mit Verzögerung, entsprechend in einem späteren Takt, weiter behandelt. Um das blockierend Verhalten der Quadranten im funktionalen Simulationsmodus zu vermeiden, kann die Arbitierungslogik auf nicht-blockierend geschaltet werden. Für die *Vaults* sowie die Speicherpartitionen sowie -bänke wird einerseits ein funktionales mit fester Latenz sowie andererseits ein speicherbandbreitenakkurates DRAM-Speichermodell auf Basis von *BOB-Sim* angeboten. Ersteres in Kombination mit dem explizit entwickelten Simulationstaktbaums ermöglicht eine passable Simulationsgeschwindigkeit für einen Multi-HMC-Betrieb.

Für nachfolgende Studien wurde eine Integration in existierende, ganzheitliche Simulatoren angestrebt, sodass das HMC-Simulationsmodell von einer größeren Nutzerbasis verwendet werden kann. Hierfür wurde beispielhaft SST genutzt. Zusätzlich wurde eine Aufzeichnungsmöglichkeit (engl.: *logging*) mit einer Datenbank-Anbindung geschaffen, damit Verläufe (*Trace*-Analyse) im Nachhinein einfacher analysiert werden können.

6.1.1 Simulationsmodell

Ein HMC-Speicherstapel besitzt bedingt durch viele unterschiedliche Komponenten – Netzwerkswitch, externe Schnittstellen, interne Kommunikation, die Quadranten, die *Vaults* sowie deren Ansteuerung als auch Partitionen und Speicherbänken – einen recht komplexen Aufbau, sodass die Wiederverwendung von frei verfügbaren Simulationswerkzeugen ein Anliegen beim Entwurf des Simulators war. Daher wurde zu Beginn auf den funktionalen Simulator *HMC-Sim* gesetzt, welcher bereits rudimentäre Unterstützung für die HMC-Spezifikation 2.0 bietet [Leidel und Chen, 2016]. Zugleich offeriert *HMC-Sim* die Möglichkeit sogenannte CMC-Instruktionen (engl.: *custom memory cube operations*, CMC) zu erstellen. Diese können als nachträglich als dynamische Bibliothek in das kompilierte *HMC-Sim*-Simulationsmodell eingeladen werden. Obgleich bereits mit vielen Aspekten ausgestattet, funktionieren jedoch die interne Adressierung der Qua-

dranten als auch der *Vaults* sowie die Multi-HMC-Umgebung innerhalb von *HMC-Sim* nicht. Auch bildet *HMC-Sim* keine FLITs ab und versendet jegliches Paket innerhalb eines Taktzyklus, wodurch nur rudimentäre funktionale Emulation statt tatsächlicher Simulation betrieben werden kann. Aus diesen Gründen wurde das erstellte HMC-Simulationsmodell hinsichtlich der Logikebene vollständig neu erarbeitet, was in einem verbesserten Simulationszeitverhalten resultierte und somit die Multi-HMC-Simulation greifbar machte. Dabei modelliert das aktuelle Modell die bis dato neueste Spezifikation in der Version 2.1. Um auch bereits auf *HMC-Sim* aufbauenden Simulationswerkzeugen eine Alternative zu bieten und obgleich die API seitens *HMC-Sim* ebenfalls nur bedingt funktionstüchtig ist, wurde ein API-*Wrapper* erstellt, welcher die API des hiermit erstellten HMC-Simulationsmodells auf die des *HMC-Sim* abbildet.

Modellierung der Speicherebenen

In Anbetracht der Tatsache, dass kein gegen ein tatsächliches Hardwaremodell validiertes HMC-Simulationsmodell existiert, trotzdem jedoch eine Taktgenauigkeit erbracht werden sollte, wurde auf zwei Ansätze innerhalb des Simulators gesetzt. Einerseits wurde ein schnell simulierendes,

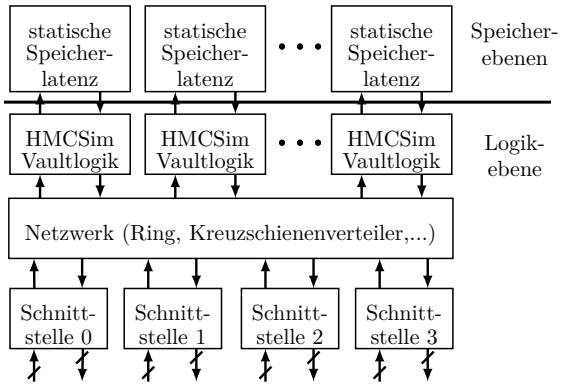


Abb. 6.1: Blockdiagramm des modellierten HMC-Simulationsmodells, welches das funktionale Speichermodell nutzt.

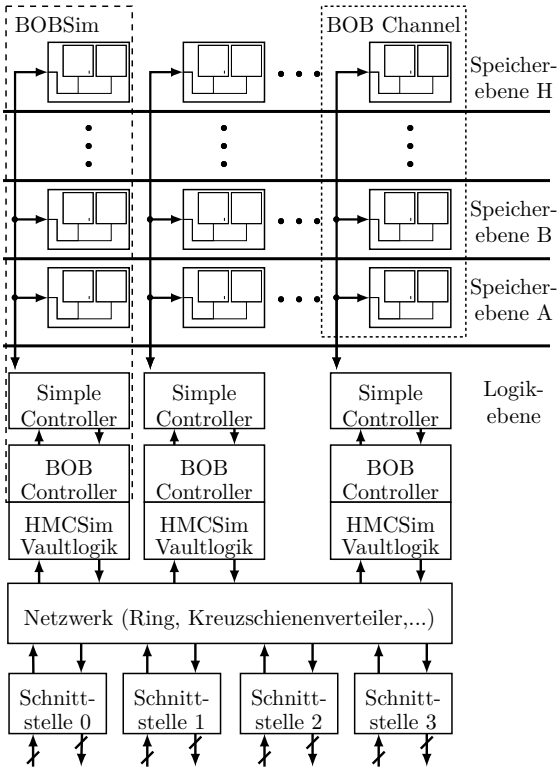


Abb. 6.2: Blockdiagramm des modellierten HMC-Simulationsmodells. Im Gegensatz zum funktionalen Speichermodell wird auf den hardwarevalidierten *BOBSim* gesetzt.

funktionales Speichermodell implementiert, welches eine fixe vorgegebene Speicherlatenz je Paket für die Speicherebene bereitstellt (Abbildung 6.1). Andererseits wurde den hardwarevalidierten Simulator *BOBSim* eingebettet, welcher die Ebenen des Speichers mit HMC-spezifischen Anpassungen taktgenau simulieren kann (Abbildung 6.2). Somit bleibt es dem Nutzer überlassen,

sich für eine schnelle jedoch ungenaue oder eine präzise, dafür langsame Speichersimulation zu entscheiden. Es kann hierbei zwischen beiden Optionen durch Neukompilieren der Simulationsbibliothek hin und her gewechselt werden, da die API gleich bleibt (Abschnitt 6.1.1).

Gerade da bei einer ganzheitlichen Entwurfsraumerkundung viele Simulationen schnell ausgeführt werden müssen, kann beim funktionalem Modell *BOBSim* per C++-Präprozessor explizit heraus kompiliert werden. Wird zugleich Rosenfelds Aussage über seine Forschung innerhalb der HMC-Thematik in Betracht gezogen, in welcher er schlussfolgert, dass unterschiedliche Speicherlatenzen (wie RAS, CAS, ...) ohnehin kaum Einfluss auf die Gesamtspeicherlatenz und auch Speicherbandbreite eines *Hybrid Memory Cube* haben [Rosenfeld, 2014], so kann bereits eine statische Latenz für viele Ansprüche, bei denen schnell Ergebnisse erzielt werden müssen, ausreichend sein.

Wird *BOBSim* aktiviert, so erzeugt die *BOBSim*-Infrastruktur einen Stapel von Speicherbänken, die Partitionen, sodass ab der *Vault*-Ansteuerung bis hin zu den Speicherbankkonflikten jegliche Details taktgenau modelliert werden können (Abbildung 6.2). Innerhalb des HMC-Simulationsmodells verbindet eine bidirektionale Schnittstelle die extrahierte *HMC-Sim-Vaultlogik* mit einem für *BOBSim* geschriebenen *Wrapper*, welcher direkt mit der in *BOBSim* vorhandenen sogenannten "einfachen Speicheransteuerung" verknüpft ist. Diese repräsentiert somit die tatsächliche *Vault*-Ansteuerung. Wurden 32 Instanzen des nicht modifizierten Simulationsmodells *BOBSim* innerhalb einer Simulation betrieben, so konnte ein signifikante Verlangsamung der Simulationsgeschwindigkeit festgestellt werden, da alle zu jedem Zyklus getaktet werden mussten. Dadurch und im Hinblick auf Multi-HMC-Konfigurationen wurde der *BOBSim*-Quelltext für das HMC-Simulationsmodells angepasst (Unterabschnitt 6.1.2). Hierzu wurde *BOBSim* auf einen einzigen Bus beschränkt, welcher die übereinander verknüpften Speicherbankebenen miteinander verbindet. Auch sind die Speicherbänke je Partition auf zwei fixiert, da dies nach den bisher eingeführten HMC-Spezifikationen den Standard bildet.

Obgleich trivial, so sollte trotzdem beachtet werden, dass es zeitliche Unterschiede zwischen funktionalem und taktgenauem Modell geben kann. Daraus resultiert, dass je Reihenfolge der zu erwartenden Rückmeldungen seitens Lade- und Speicheroperationen variieren kann, da *BOBSim* intern die Rückmeldung von Speicheroperationen nicht in die zurückführende Warteschlange der Ladeoperationen sondern in eine dafür eigenständige Warteschlange einbringt. Entsprechend können Rückmeldungen für Speicherinstruktionen Ladeinstruktionen überspringen. Um *BOBSim* nicht nur an die Architektur eines HMC-Speichers anzupassen, sondern auch hinsichtlich der Latenzen, wurde das Modellierungswerkzeug *CACTI 7.0-prerelease* zu deren Generierung genutzt. Somit ist es dem HMC-Simulationsmodell möglich, entweder die Speicherlatenzen von *DDR3*-DRAM oder von *CACTIs* HBM zu nutzen, jedoch steht es dem Nutzer frei weitere Konfiguration in Erwägung zu ziehen.

Im Rückblick lässt sich für eine Entwurfsraumexploration das HMC-Simulationsmodell wie folgt betreiben: Zuerst können eine Vielzahl an potentiellen HMC-Konfigurationen mit dem funktionalen Modus evaluiert werden. Wird eine Konfiguration als wertvoll erachtet, kann transparent auf annähernd taktgenau umgeschaltet werden, wodurch eine tiefe Analyse mit Hilfe von *BOBSim* erzielt werden kann.

Modellierung der Logikebene

Kernstück der Logikebene bilden die vier Quadranten, welche jeweils sieben unabhängig voneinander agierende, interne Speicherstapel, den sogenannten *Vaults* miteinander verbindet. Da nicht definiert wird davon ausgegangen, dass ein Quadrant jeweils eine externe Schnittstelle bedienen kann, sind doch bis zu vier externe Schnittstellen vorgesehen und es existieren ebenfalls vier Quadranten. Die HMC-Spezifikation trifft hierzu keine Aussage, jedoch kann das entwickelte HMC-Simulationsmodell flexibel auf neuere Spezifikationen angepasst werden. Selbiges trifft auf die interne Vernetzung zu, sodass nicht definiert ist wie die Logikebene im Detail zwischen einzelnen Quadranten sowie zwischen Quadrant und *Vault* vernetzt ist. Daher wurden ein parametrisierbarer Kreuzschienenverteiler (engl.: *crossbar*) und ein ebenfalls anpassbarer Ringbus entwickelt, welche zumindest eine interne Vernetzung ermöglichen. Sofern an eines der beiden vernetzt, erlauben die Quadranten d.h. eine blockierende oder nicht-blockierende Terminierung von eingehenden, elementaren Datenpaketen. Diese wiederum werden entsprechend weiter an die *Vaults* sowie die externen Schnittstellen weitergeleitet. Gerade dies ist wiederum in der HMC-Spezifikation definiert, welche aussagt, dass alle Zugriffe auf einen *Vault* innerhalb eines Quadranten innerhalb eines Taktes geschehen sollen.

Eine *Round-Robin*-Strategie wurde in die Logik der Quadranten integriert, um die Arbitrierung von mehreren internen Schnittstellen kommenden vom Kreuzschienenverteiler oder Ringbus zu ermöglichen. Zeitgleich kann dadurch eine Simulation von Anschlusskonflikten im blockierenden Modus ermöglicht werden, welcher ja nach Konfiguration aktiviert werden kann. Da die HMC-Spezifikation die Details der Logikebene nicht festlegt, sind beide Implementierungen valide; als gekapselte C++-Klassen implementiert lassen sie sich für zukünftige Studien einfach ersetzen.

Für die Quadranten ist anzumerken, dass diese bei gegebenen speicherintegrierten Verarbeitungseinheiten weitere Schnittstellen für die Interaktion mit dem Speicherstapel bieten müssen. Daher kann die Menge an internen Endstellen des jeweiligen Quadranten vor Kompilierung mit Hilfe eines Parameters eingestellt werden. Zugleich werden die Endstellen innerhalb des jeweiligen C++-Konstruktors miteinander vernetzt, wodurch dies erst bei Starten eines HMC-Simulationsmodells geschieht. Zeitgleich bietet es die Möglichkeit, weitere interne Schnittstellen an speicherintegrierte Verarbeitungseinheiten anzubinden, wodurch diese das interne Netzwerk der Logikebene verwenden können. Im Gegensatz zur tatsächlichen HMC-Hardware besteht keine Restriktion hinsichtlich Datenpaket-Laufrichtung, wodurch das HMC-Simulationsmodell als Testplattform für von speicherintegrierten Verarbeitungseinheiten erstellte Fernzugriffe genutzt werden kann.

Um eine angemessene Zeitmodellierung zu erlauben, wurde eine getaktete, vollduplexfähige Verbindung zwischen allen Quadranten, einem jeweiligen *Vault* und einem Quadranten sowie für die externen Schnittstellen erstellt (Abbildung 6.3). Jede der Verbindungen berücksichtigt beim Transfer die Paketlänge in FLITs und berechnet anhand der zuvor gesetzten Bitrate also Taktfrequenz und Bitbreite die benötigte Zeit für das jeweilige Paket. Zugleich wurden anhand der Hardwarespezifikation entsprechende Sende- und Empfangspuffer implementiert, wobei der Sendepuffer ein eingehendes Datenpaket in die einzelnen FLITs zerlegt, welche dann über die Verbindung versendet werden. Der Empfangspuffer hingegen setzt die einzelnen FLITs wieder in ein versendetes Paket zusammen. Ein HMC-Speicherstapel sieht hierfür eine zyklische Redundanzprüfung (engl.: *CRC checksum*) vor, anhand welcher erkannt werden kann, ob das entspre-

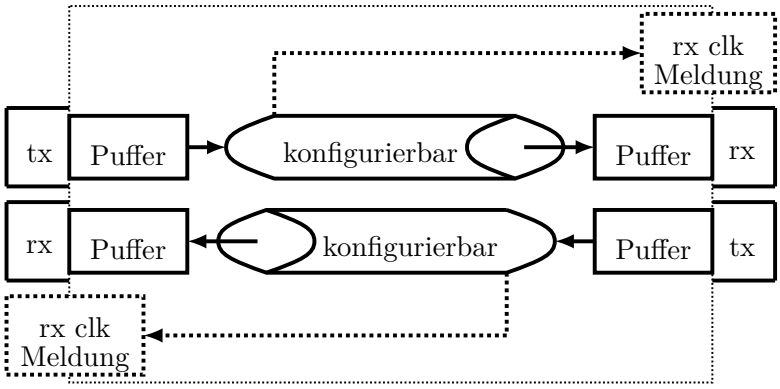


Abb. 6.3: Schema zweier bidirektionaler Vollduplex-FIFOs mit Sende- (tx) und Empfangspuffern (rx). Beide FIFOs sind hinsichtlich der Bitrate und Bitbreite flexibel per Parameter konfigurierbar. Zugleich bieten beide Empfangsrichtungen eine Benachrichtungseinrichtung, welche bei Empfang eine zuvor registrierte Funktion aufruft. Solche Benachrichtigung setzt meist ein Bit, um bei der späteren Simulationstaktung als Hinweis für Taktbedarf zu fungieren.

chende Paket korrekt ist oder nicht. Diese ist innerhalb des HMC-Simulationsmodells ebenfalls implementiert worden, wird jedoch aus Gründen der Simulationsgeschwindigkeit standardmäßig deaktiviert. Sofern jedoch das Verhalten eines HMC bei Fehlerinjektion betrachtet werden soll, kann der Mechanismus entsprechend aktiviert werden.

Die zur beschriebenen Verbindungen werden nicht nur als externe Schnittstellen zwischen HMCs oder zwischen HMC und einem Host (wie z.B. einem Prozessor), sondern zeitgleich aufgrund ihrer Flexibilität für HMC-interne Verbindungen genutzt. Somit lässt sich auch eine vorab definierte Paketstruktur innerhalb des Simulationsmodells leichter handhaben. Diese führt innerhalb eines Pakets die tatsächlichen Daten üblicher Weiße mit sich. Dies ist jedoch bei Speichermodellen, welche auf hohe Simulationsgeschwindigkeit optimiert sind nicht vorgesehen, sodass auch innerhalb diesem Simulationsmodells das Mitführen und Schreiben von Daten optional ist. Sofern die Daten nicht durch das HMC-Simulationsmodell traversieren, liegt es am Nutzer eine externe Möglichkeit zu schaffen Daten tatsächlich an eine Speicherposition zu schreiben. Entsprechend fungiert das Simulationsmodell rein zur Generierung des Taktverhaltens und zur Evalutionsmöglichkeit der Pakettraversierung, jedoch bietet das Nichtschreiben und Nichtmitführen von Daten eine signifikante Beschleunigung der Simulationsgeschwindigkeit. Ob nun die Daten durch das Modell traversieren und in den Speicher geschrieben werden sollen, oder ob nur ein zeitliches Verhalten mit Hilfe von datenleeren und nicht zu schreibenden Paketen genutzt werden soll, steht dem Nutzer anhand von einer Auswahlmöglichkeit frei. Es soll nicht unerwähnt bleiben, dass das HMC-Simulationsmodell zwar die HMC-Spezifikation nahtlos implementiert, jedoch die zu erwartenden Speicherbandbreiten als auch Speicherlatenzen eine optimistische Obergrenze eines tatsächlichen HMCs darstellen. Dies resultiert aus der nicht vollständigen Spezifikation im Bereich von kritischen Komponenten wie etwa der Logikebene.

Obgleich die Aufzeichnung eines Simulationsverlaufs einen zentralen Aspekt der Simulation darstellt, werden häufig die erstellten Daten auf die Konsole (`stdout`) ausgeschrieben oder aber in eine Datei. Beide Fälle erschweren eine nachfolgende Analyse, da die Daten zuallererst wieder aufgearbeitet werden müssen. Mit einem standardisierten Berichterstattungs- und Aufzeichnungsframework zeigten Meyer et al. auf, dass das direkte Mitschreiben der Daten in eine Datenbank nur unwesentlich zur Simulationsgeschwindigkeit beiträgt [Meyer et al., 2016].

Gleichzeitig lässt dies den Schritt der Aufarbeitung der Daten wegfallen, wodurch Zeit eingespart werden kann. Diesem Konzept folgend, wurde eine generische Schnittstelle innerhalb des Simulators errichtet, welche je nach Konfiguration die Daten auf die Konsole, in die Datenbank *SQLite* oder die Datenbank *PostgreSQL* schreibt. Zusätzlich können weitere Backends erstellt werden, welche die generische Schnittstelle adaptieren.

Im Falle der Datenbank *SQLite* wurden zur Beschleunigung der Simulationsausführung eine Vermeidung von schwergewichtigen *SQL*-Einfügungen vermieden, indem ASCII-*SQL*-Anweisungen vorab zu Binär-*SQL*-Anweisungen kompiliert werden. Entsprechend muss während der Laufzeit keine *Just-In-Time*-Kompilierung der *SQL*-Anweisung mehr getätigt werden. Zusätzlich wird anstatt auf das Dateisystem zuerst in eine Datei im Hauptspeicher geschrieben, welche erst nach Beendigung der Simulation wiederum in das Dateisystem geschrieben wird. Nicht implementiert, jedoch von Meyer et al. aufgezeigt, könnte sowohl ein zur Laufzeit stattfindendes Vorfiltern eingebracht als auch ein zweiter Prozess genutzt werden, welcher das Filtern übernimmt [Meyer et al., 2016]. Aktuell hingegen schneidet jede interne und externe Verbindung während der Simulation ein- und ausgehenden FLITs vollständig mit.

Software-API

Das funktionale Simulationsmodell *HMC-Sim* bietet bereits eine API, welche zur Anbindung an weitere Simulatoren genutzt werden könnte. Jedoch bietet dieses hinsichtlich der Anpassung von Bitrate und Bitbreite aufgrund des funktionalen Modells einen geringeren Funktionsumfang. Im Gegensatz dazu bietet das errichtete HMC-Simulationsmodell die Möglichkeit auch zur taktgenauen Simulation und muss entsprechend Parameter zur Anpassung bereitstellen. Inspiriert

Auflistung 6.1: Software-API zur Anbindung des HMC-Simulationsmodells.

```
hmc_sim( unsigned num_cubes, unsigned num_slids,
         unsigned num_links, unsigned capacity,
         unsigned quadbus_bitwidth, float quadbus_bitrate );

hmc_jtag* hmc_get_jtag_interface( unsigned cubeId );

bool hmc_set_link_config( unsigned src_cubeId,
                          unsigned src_linkId,
                          unsigned dst_cubeId,
                          unsigned dst_linkId,
                          unsigned bitwidth,
                          float bitrate );
hmc_notify* hmc_define_slid( unsigned slidId,
                             unsigned cubeId,
                             unsigned linkId,
                             unsigned lanes /* = Bitbreite */,
                             float bitrate );

bool hmc_send_pkt( unsigned slidId, char *pkt );
bool hmc_recv_pkt( unsigned slidId, char *pkt );
```

durch die tatsächliche Anbindung eines HMC-Speicherstapels wurde eine eigenständige und auf

das Minimum beschränkte Software-API implementiert (Aufl. 6.1). Per C++-Konstruktor wird die tatsächliche Simulationsumgebung aufgesetzt, bei welcher die Anzahl der zu verwendenden Speicherstapel, die Anzahl an Endverbindungspunkten (engl.: *source-link identifier*, SLID), die Anzahl der externen Schnittstellen je HMC sowie deren Speicherkapazität anzugeben ist. Prinzipiell werden alle externen Schnittstellen unter dem Titel *Link* geführt, wobei Endverbindungspunkte, welche den Ausgang aus dem HMC-Netzwerk bilden, als SLID bezeichnet werden. Nicht unter einem spezifischen Begriff gelistet, jedoch für die Verbindungsgeschwindigkeit zwischen Quadranten von Belang, müssen zeitgleich die Geschwindigkeiten also Bitrate und Bitbreite derer an dieser Stelle definiert werden.

Ist dies geschehen, muss im Falle ohne Umgebungsvariable im Quelltext die jeweiligen *Links* zwischen verschiedenen HMCs und die SLID zwischen HMC und Treiber (z.B. Prozessor) definiert werden. Dies geschieht mit `hmc_set_link_config()` und `hmc_define_slid()`, bei welchen jeweils ein gewisser HMC spezifiziert und zeitgleich die Geschwindigkeit definiert wird. Für den Fall, dass multiple Vernetzungskonfiguration schnell evaluiert werden sollen, bietet sich eine Umgebungsvariable an, welche auf eine im *DOT*-Format vorliegende Datei verweisen muss. *DOT* selbst ist eine Graph-Beschreibungssprache welche die Möglichkeit bietet, gerichtete und ungerichtete Graphen als ASCII-Format darzustellen. Entsprechend kann eine derartige Datei dynamisch auf eine gewünschte Konfiguration gesetzt werden, welche innerhalb des Simulators als Vernetzung zwischen HMCs und Treiber eingespielt wird. Hierbei kann ebenfalls die Geschwindigkeit also Bitrate und Bitbreite spezifiziert werden. Daraufhin kann bereits die Sendemethode `hmc_send_pkt()` des HMC-Simulationsmodells genutzt werden um Pakete in das Geflecht aus HMCs einzubringen (Aufl. 6.2). Zusätzlich kann mit `hmc_recv_pkt()` der Simulator nach einem ausgehenden Paket angefragt werden. Da beide Funktionen bedingt durch internes Kopieren des Paketes Rechenaufwand bedingen, dies jedoch nicht für die Simulationsgeschwindigkeit vorteilig ist, bietet die Funktion `hmc_define_slid()` eine Benachrichtigungsstruktur für die API-Methode `hmc_recv_pkt()` an. Im Falle der *DOT*-Konfiguration wird explizit die Methode `hmc_notify* hmc_get_slid_notify(void)` angeboten, welche dem Treiber wie z.B. einem Prozessormodell ebenfalls die Möglichkeit für eine schnelle Evaluierung der eingehenden Warteschlangen bietet. Die Struktur `hmc_notify` kann daraufhin bei jedem Zeitschritt auf ein gesetztes Bit hin überprüft werden (Aufl. 6.2). Ist dies der Fall, so befindet sich ein fertig übermitteltes Paket in der Ausgangsverbindung.

6.1.2 Beschleunigung der Simulationsgeschwindigkeit

Aufgrund des internen Aufbaus einer HMC-Speicherarchitektur kann eine Baumstruktur ausgemacht werden. Diese spannt sich von den Quadranten bis hin zur einzelnen Bank. Wird von einem kanonischen Simulator ausgegangen, so stellt meist die Wurzel eine Simulationsstruktur dar, welche in diesem Fall mehrere Speicherstapel als Kindknoten beherbergt. Diese fächern sich weiter in die jeweiligen *Vaults* und daraufhin weiter in die Partitionen und Speicherbänke auf. Solch ein kanonischer Simulator würde bei jedem Zeitschritt jeweils mit Schleifen über die HMCs, jeder HMC mit Schleifen über die Quadranten etc. iterieren, um schließlich bis in die tiefste Ebene vorzudringen. Jedoch wird nicht zu jedem Takt jeglicher Knoten innerhalb des Baumes den Bedarf haben, getaktet zu werden. Diesen Effekt machen sich gängige Simulatoren zunutze, indem eine Verzögerungsmöglichkeit in den Simulationsquelltext eingeführt wird, wodurch mit Hilfe einer globalen Terminierung die weitere Ausführung um eine spezifische Zeit verzögert wird.

Auflistung 6.2: Beschleunigte Überprüfung auf eingehende Netzwerkpakete.

```
// Einmalig:
  unsigned slids = ...;
  hmc_sim sim( ..., slids, ...);
  hmc_notify* slidnotify = sim.hmc_get_slid_notify();
  char packet[17*FLIT_WIDTH / (sizeof(char)*8)];
  char retpacket[17*FLIT_WIDTH / (sizeof(char)*8)];

// Bei jedem Takt:
  // Ausgehende Pakete:
  unsigned slidId = ...;
  if(!sim.hmc_send_pkt(slidId, packet)) {
    // Fehlschlag beim Versenden eines Pakets.
  }

  // Eingehende Pakete:
  unsigned slid_maske = slidnotify->get_notification();
  if(slid_maske) {
    for(unsigned i = 0; i < slids; i++)
      if(slid_maske & (0x1 << i)) {
        sim.hmc_recv_pkt(i, retpacket);
        // Behandlung des eingehenden Pakets.
      }
  }

  // Takten:
  sim.clock();
```

Dies führt jedoch meist zu Mehraufwand durch signifikante Funktionsaufrufe sowie Speichern des Registersatzes, welcher zur Weiterführung später wiederum eingeladen wird.

Zur Ermöglichung einer leichtgewichtigen Indikation einer Taktnotwendigkeit wurde die Baumstruktur des HMC-Speicherarchitektur ausgenutzt, indem jegliche interne als auch externe vollduplexfähige Verbindungen um eine Registriermöglichkeit für einen optimierten Taktbaum erweitert wird (Abbildung 6.3). Beide Empfangsenden bieten dadurch solch eine Registriermöglichkeit. Wird nach erfolgreicher Registrierung und bei laufender Simulation ein Datenpaket als vollständig übertragen identifiziert, so wird diese vordefinierte Methode zur Empfangsquittierung ausgeführt. Das Netzwerkpaket selbst bleibt jedoch noch innerhalb der Verbindung und wird erst mit dem kommenden Takt ausgelesen und nachfolgend behandelt.

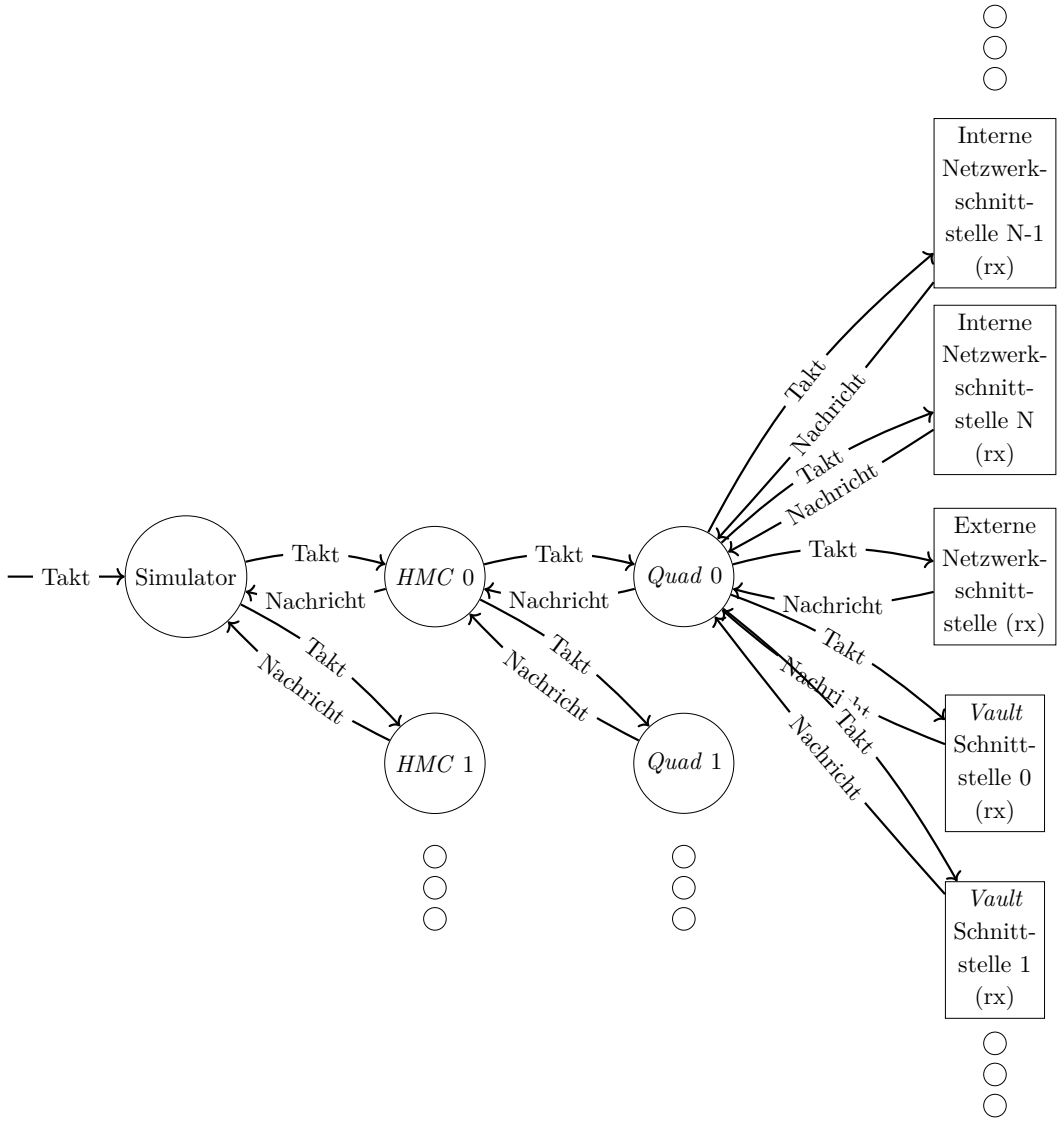


Abb. 6.4: Schema des für das HMC-Simulationsmodell entwickelten optimierten Simulationstaktbaums.

Durch die Vernetzung der übergeordneten Taktbäume kann eine Baumstruktur aufgebaut werden, die jeweils eine Ebene höher und bis zur Wurzel über den Empfang informiert (Abbildung 6.4). Auf jeder Ebene wird für das benachrichtigende Kind ein Bit zur zukünftigen

Dekodierung dessen gesetzt. Eine zukünftige Taktung des kompletten HMC-Simulationsmodells geschieht wiederum von oben nach unten. Dies bedeutet, dass jede Ebene dabei vergleicht, ob eines der Bits innerhalb einer Bitmaske für die jeweiligen "Kindkomponenten" gesetzt ist. Sofern dies zutreffen sollte, wird die Komponente getaktet, ansonsten nicht. Gerade der letzte Fall, d.h. das Nicht-Takten, verhilft dabei zur Beschleunigung der Simulation. Sofern die tiefste Ebene einen Taktbedarf nach oben eskaliert, so findet bedingt durch die Baumstruktur auf jeder Ebene eine Ausdünnung statt. Deshalb wird bei einer Eskalation nach oben auf jeder Ebene untersucht, ob nicht bereits ein Bit innerhalb der Bitmaske für das jeweilige Kind gesetzt ist. Dadurch kann in gewissen Fällen eine Traversierung des Baums bis zur Spitze vermieden werden, sodass die neu eingeführte Beschleunigungsstruktur nicht unnötig Rechenzeit konsumieren. Wie bereits angedeutet, findet bei jedem Simulationstakt ein Takten von oben nach unten innerhalb des Baums statt. Dabei wird nun auf jeder Ebene die notwendige Bitmaske herangezogen, um zu erkennen, ob das entsprechende Kind Taktbedarf angemeldet hat.

Da jegliche Verbindung innerhalb des HMC-Simulationsmodells solch ein optimierten Taktbaum-Endpunkt besitzt und auch der Treiber – welcher als *main()*-Quelltext oder als Prozessorsimulator vertreten sein kann – davon profitieren soll, übergibt jeder SLID-Endpunkt eine dafür notwendige Struktur an den Treiber (Aufl. 6.1). Somit kann dieser leichtgewichtig erkennen, ob weitere Pakete vom HMC-Simulationsmodell mit Hilfe der schwergewichtigen `hmc_recv_pkt()`-API-Funktion empfangen werden können (siehe Aufl. 6.2 mit Fokus auf die eingehenden Pakete). Obgleich die Benachrichtigungsbaumstruktur innerhalb des HMC-Simulationsmodells im schlimmsten Fall einen Mehraufwand darstellt d.h. sofern alle Verbindungen und alle HMC-Speicher getaktet werden müssen, kann davon ausgegangen werden, dass im praktischen Fall die Lade-/Speicherwarteschlange eines oder mehrere Prozessoren eine Limitierung der im HMC befindlichen Pakete und dadurch FLITs generiert. Bedingt durch typischerweise kleine Lade-/Speicherwarteschlangen werden zu einem gegebenen Zeitpunkt nicht alle Verbindungen des HMC-Simulationsmodells getaktet werden müssen.

Im Falle, dass *BOBSim* als Speichermodell genutzt wird, ist bedingt durch die internen Zustände dessen mehr Simulationsaufwand notwendig. Wurde eine Lade- oder Speicheroperation von *BOBSim* als vollständig bearbeitet quittiert, so wird die entsprechende *BOBSim*-Instanz bis zu dem Neueinlesen der im DRAM-Speicher gehaltenen Daten weiter getaktet. Sollten darauf folgende Lade- und Speicheroperationen erscheinen, so wäre *BOBSim* trotz alledem in einem synchronen Zustand, sofern er zu einem taktenden Zeitpunkt eine neue Operation erhält. Ist dies nicht der Fall, so kann davon ausgegangen werden, dass *BOBSim* mit hoher Wahrscheinlichkeit seit geraumer Zeit keine weiteren Operationen erhalten hat, wodurch sich die internen Zustände auf *Refreshing* eingependelt haben. In diesem Fall wäre für eine neue Operation exakt der Zustand nach einem Neueinlesen vorgesehen. Obgleich dieser Zustand bei einem permanenten Takten nicht zwingend derselbe ist wie mit diesem Ansatz verfolgt, so ist bei Verwendung dessen ohnehin davon auszugehen, dass primär Simulationsgeschwindigkeit im Gegensatz zu Simulationsgenauigkeit als Ziel gesetzt wird. Entsprechend kommt eine leichte Divergenz zustande, wird doch die Zeit bis zum nächsten *Refresh* verschoben. Wurde hingegen das funktionale Speichermodell gewählt, so werden die *Vaults* ab dem Zeitpunkt, an welchem keine weiteren Pakete innerhalb des entsprechenden *Vaults* vorhanden, sind nicht mehr getaktet.

Da auf Simulationsgeschwindigkeit ausgelegt, ist die ebenfalls implementiere, für Fehlerinjektionsstudien nutzbare, zyklische Redundanzprüfung (CRC) standardmäßig deaktiviert. Ein weiterer, einfacherer Aspekt war die Vermeidung von dynamischer Allokation zur Laufzeit. Dies

wurde innerhalb des vollständigen Simulators vermieden, sodass zum aktuellen Zeitpunkt der Simulator ausschließlich bei Lade- sowie Speicheranfragen kleine Mengen an Speicher dynamisch allokiert. Um dies zu beschleunigen wurde auf Googles im Hochgeschwindigkeitsberechnen bekannte Bibliothek *TCMalloc* (*Thread-Caching Malloc*), welche innerhalb der *perftools*-Sammlung verfügbar ist, zurückgegriffen [Lee et al., 2014]. Zu guter Letzt wurde mit dem Linux-Programm *perf* der Simulator auf Geschwindigkeitsverluste hin überprüft und optimiert [Carvalho de Melo, 2010].

6.1.3 Multi-HMC-Konfigurationen

Im Gegensatz zu bekannten, planaren Speicherarchitekturen bietet die HMC-Speicherarchitektur als Schlüsseltechnologie eine Möglichkeit zur Vernetzung multipler HMC-Speicherstapel basierend auf Paketen. Angesprochene HMC-Simulationsmodelle bieten hierfür keine Unterstützung (Abschnitt 5.1.2). Die Simulation von multiplen HMCs stellt einen signifikanten Mehraufwand dar, müssen doch alle jeweils getaktet werden, was innerhalb eines Simulators bedingt durch die Garantie eines konsistenten Takts zu sequentieller, entsprechend langsamer Ausführung führt. Entsprechend nutzt die Multi-HMC-Simulation die gewonnen Simulationsgeschwindigkeit aus Unterabschnitt 6.1.2.

Die Vernetzung definiert mit Version 2.1 der HMC-Spezifikation ein Vorwärtsprotokoll je Speicherschnittstelle, sodass sich ausschließlich sternförmige oder hintereinander verkettete Netzwerke aus HMC-Speicherstapeln bilden lassen, folglich allerdings keine Ringstrukturen. Da das entwickelte HMC-Simulationsmodell für Forschung ausgelegt ist, kann diese Beschränkung sofern gewünscht aktiviert werden, jedoch ist dies standardmäßig deaktiviert. Um das Netzwerk zu ermöglichen, wurde ein Routing-Algorithmus benötigt, welcher einerseits die kürzeste Distanz und andererseits die zu einem Zeitpunkt noch nicht benutzten Schnittstellen findet und anhand dessen gewichtet. Zu diesem Zweck wurde ein Routing-Algorithmus entworfen, welcher einerseits bei der Netzwerkinitialisierung eine Routing-Tabelle für die wenigstens Sprünge (engl.: *hops*) zwischen Start- und Ziel-HMC berechnet und andererseits zur Laufzeit die Schnittstellen beim Routen in Erwägung zieht. Da der Routing-Algorithmus nicht durch die Spezifikation definiert ist, wurde dieser in einer C++-Klasse gekapselt, wodurch unterschiedlichste Netzwerkalgorithmen für zukünftige Forschung einfach adaptiert werden können. Da gekapselt, jedoch innerhalb des Modells, ist der Routing-Algorithmus statisch gewählt, jedoch muss davon ausgegangen werden, dass in einer realen Multi-HMC-Konfiguration eine solche Routing-Information per JTAG-, I2C- oder SPI-Schnittstelle in den jeweiligen Speicherstapel geladen wird. Dies ist jedoch nicht in der Spezifikation verankert.

Bereits in Abschnitt 6.1.1 angedeutet, wird eine Umgebungsvariable für das dynamische Laden von Routing-Informationen zwischen verschiedenen HMCs sowie HMC und Treiber (z.B. Prozessor) geboten. Hintergrund ist eine leichtere Evaluierung von unterschiedlichen Netzwerktopologien, ohne dass der Quelltext angepasst werden muss. Hierzu wird auf die Graphbeschreibungssprache *DOT* zurückgegriffen, welche komplexe gerichtete und ungerichtete Graph-Strukturen erzeugen kann. Für das HMC-Simulationsmodell wird auf ungerichtete Graphen zurückgegriffen, wobei ein Beispiel in Aufl. 6.3 gegeben ist. Wie zu erkennen, sind Schnittstellen eines HMC-Speichers sowie die spezifischen SLID-Endpunkte innerhalb des Graphen als Ecken definiert, während die Vollduplex-Verbindungen als Kanten repräsentiert werden. Kanten können dabei mit Bitbreite sowie Bitrate annotiert werden, welche der HMC-Spezifikation entsprechen müssen.

Auflistung 6.3: Ein mit *DOT* beschriebener ungerichteter Graph, welcher eine Multi-HMC-Konfiguration abbildet. Der Prozessor bietet acht SLID-Einspeispunkte, wohingegen jeder HMC mit vier Vollduplex-Verbindungen (*Links*) ausgestattet ist. Die jeweilige Vollduplex-Verbindung wird über zwei Minuszeichen zwischen zwei HMC-Speichern oder HMC-Speicher und Prozessor errichtet. Zugleich wurde jede Vollduplex-Verbindung auf die maximal zulässige Geschwindigkeit konfiguriert, d.h. eine Busbreite von 16 Bit und eine Bitrate von 30,0 GB/s.

```
graph G {
 node[style=rounded, shape=record];
 HOST[label="{HOST|{<slid0>slid0|<slid1>slid1|<slid2>slid2|<slid3>slid3
    }}"];
 HMC0[label="{{<link0>link0||<link1>link1}|HMC0|{<link2>link2||<link3>
    link3}}"];
 HMC1[label="{{<link0>link0||<link1>link1}|HMC1|{<link2>link2||<link3>
    link3}}"];

 HOST:slid0 -- HMC1:link2 [label="16,BR30"];
 HOST:slid1 -- HMC0:link0 [label="16,BR30"];

 HOST:slid2 -- HMC0:link1 [label="16,BR30"];
 HOST:slid3 -- HMC1:link3 [label="16,BR30"];

 HMC0:link2 -- HMC1:link0 [label="16,BR30"];
 HMC0:link3 -- HMC1:link1 [label="16,BR30"];
}
```

Zugleich steht das mächtige frei verfügbare Graph-Visualisierungswerkzeug *Graphviz* sowie die C++-Bibliothek *Boost* zur Verfügung [Schling, 2011]. *Graphviz* erlaubt es *DOT*-Dateien zu

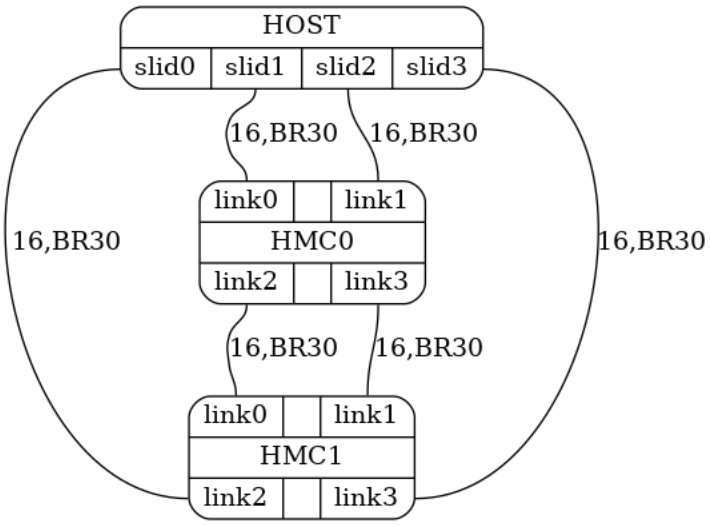


Abb. 6.5: Schema einer im HMC-Simulatormodell unterstützten Multi-HMC-Konfiguration, welche mit Hilfe von *Graphviz* und einer *DOT*-Konfiguration erzeugt wurde (Aufl. 6.3). Solch Konfiguration entspricht nicht der offiziellen HMC-Spezifikation, ist jedoch für Forschungszwecke möglich.

einem *PNG*- oder *PDF*-Bild zu visualisieren [Ellson et al., 2001] (Abbildung 6.5). Die C++-Bibliothek *Boost* kommt hingegen innerhalb des HMC-Simulationsmodell nach Abfrage der exportierten Umgebungsvariable zum Einsatz, welche wiederum den Ort der zu parsenden *DOT*-Datei angibt. Nach Parsen der Datei wird die Vernetzung im HMC-Simulationsmodell *Links* zwischen HMCs untereinander sowie SLIDs zwischen HMC und Treiber, wie z.B. einem Prozessor, dynamisch erstellt. Auch eine Mischform aus *DOT*-Graphen und statischen API-Aufrufen zur Konfiguration der *Links* und SLID ist denkbar.

Für zukünftige Entwurfsraumexplorationen könnten in einer Datenbank befindliche, extrahierte Verbindungsstatistiken zurück in eine *DOT*-Datei und somit in den *DOT*-Graphen geschrieben werden, wodurch eine einfache Visualisierung für den Entwickler stattfinden kann.

6.1.4 *Wrapper* für Structural Simulation Toolkit (SST)

Auf Basis der Integration von *HMC-Sim* in die Simulationswerkzeugsammlung SST [Rodrigues et al., 2011] wurde ein sogenannter *Wrapper* implementiert, welcher die *HMC-Sim* Programmierschnittstellen (API) in der Version 2.0 nachahmt. Da der *Wrapper* leicht anpassbar und aktuell binärkompatibel zu *HMC-Sim* ist, konzentriert sich eine zukünftige Pflege ausschließlich auf den *Wrapper* für SST, nicht jedoch das entwickelte HMC-Simulationsmodell. Dies verhilft zu einer einfachen Verwendbarkeit des erbrachten Simulationsmodells und einer einfacheren nachfolgenden Wartung.

Die architektonische Nähe beider Simulatoren bedingt durch das gemeinsame Ziel, der Simulation eines HMC, verhilft dem entwickelten HMC-Simulationsmodell dazu, dass multiple Schnittstellenfunktionen nahezu direkt übersetzt werden konnten (Tabelle 6.1). Entsprechend

HMC-Sim Programmierschnittstelle	Funktionsaufruf im HMC-Simulationsmodell
`hmcsim_init(hmcsim_t *hmc, ...)`	`hmcsim = new hmc_sim(...)`
`hmcsim_free(hmcsim_t *hmc)`	`delete hmcsim`
`hmcsim_link_config(hmcsim_t *hmc, ...)`	`hmcsim->hmc_define_slid(...)` / `hmcsim->hmc_set_link_config(...)`
`hmcsim_build_memrequest(hmcsim_t *hmc, ...)`	`hmcsim->hmc_encode_pkt(...)`
`hmcsim_decode_memresponse(hmcsim_t *hmc, ...)`	`hmcsim->hmc_decode_pkt(...)`
`hmcsim_send(hmcsim_t *hmc, ...)`	`hmcsim->hmc_send_pkt(...)`
`hmcsim_recv(hmcsim_t *hmc, ...)`	`hmcsim->hmc_recv_pkt(...)`
`hmcsim_clock(hmcsim_t *hmc)`	`hmcsim->clock()`
`hmcsim_get_clock(hmcsim_t *hmc, ...)`	`hmcsim->hmc_get_clock()`

Tab. 6.1: Abbildung einiger elementarer *HMC-Sim* v2.0 unterstützten SST-API-Schnittstellen auf Funktionen des HMC-Simulationsmodells.

ist nur ein weiterer Funktionsaufruf mehr zu tätigen. Einzig bei der externen Vernetzung eines HMC bietet die *HMC-Sim* API keine Differenzierung, d.h. ob es sich bei der simulierten Hardwareschnittstelle um einen Verbindung zwischen HMC und Prozessor, oder aber zwischen zwei HMCs handelt. Dieser Unterschied ist jedoch bei einer Multi-HMC-Simulation von Nöten, da hierbei anhand einer SLID erkannt wird, ob die jeweilige Schnittstelle als Ein- oder Ausgang in das HMC-Netzwerk (seitens eines Prozessors) angesehen werden kann. Der *Wrapper* stellt wiederum beim Erstellen der jeweiligen Verbindungen eine korrekte Vernetzung des HMC-Simulationsmodells sicher. Nicht unerwähnt soll das Laden einer Netzwerkkonfiguration per Umgebungsvariable bleiben (Unterabschnitt 6.1.3), welche im Gegensatz zur aktuell statischen Vernetzung für die Zwecke SSTs mehr geeignet wäre.

6.2 GPGPU-Rechenbeschleuniger

Das Projekt *NYUZI* mit der gleichnamigen Instruktionssatzarchitektur bietet einerseits eine frei zugänglich RTL-Implementierung sowie einen ebenfalls frei verfügbaren C-basierten Emulator. Während letzterer rein funktional den Instruktionssatz ausführen kann, kann die in SystemVerilog beschriebene RTL-Implementierung einerseits innerhalb eines FPGAs lauffähig gemacht oder andererseits mit Hilfe des Werkzeugs Verilator in C++ übersetzt und später mit regulären Compilern kompiliert werden. Dadurch kann die RTL-Implementierung auf gängiger Hardware als Modell simuliert werden. Letzteres Konzept bietet im Gegensatz zur nochmaligen Entwicklung eines Simulators den Vorteil, dass keine Approximation des Modells vorgenommen werden muss, weil jederzeit die aktuelle Hardware simuliert wird. Dies führt jedoch zu Lasten der Ausführungsgeschwindigkeit, da jegliches Detail der Architektur simuliert werden muss. Zur Vermeidung dessen und zur Ermöglichung von schnellen Simulationsausführungen wurde ein C++-Modell des GPGPU-Rechenbeschleunigers entwickelt.

6.2.1 Struktureller Aufbau der GPGPU

Zum aktuellen Zeitpunkt ermöglicht der *NYUZI*-GPGPU-Rechenbeschleuniger ein flexibles Anpassen jeglicher Ressourcen in etwa des Instruktionssatz, der Menge an Rechenkernen, Hardware-

Threads, Länge der Verarbeitungspipeline und Vektorregisterbreite. Grundsätzlich baut die Architektur noch auf einer 32 Bit breiten Little-Endian-ISA auf, welche in der Standardeinstellung einen Rechenkern mit vier Hardware-Threads betreibt, was jedoch gleichsam aufgrund der Offenheit angepasst werden kann. Jeder Rechenkern nutzt die *Harvard-Architektur* für die erste Cachehierarchie, sodass je 16 KB an Instruktions- und Datencache zur Verfügung gestellt werden. Demgegenüber steht ein L2-Cache, welcher 128 KB für alle Rechenkerne bereitstellt. Alle Caches sind speicherkohärent. Der aktuellen Breite der ISA geschuldet bietet die aktuelle GPU-Architektur jedem Hardware-Thread 32 Skalarregister, 32 Vektorregister mit 512 Bit Breite und ein dediziertes Befehlszählerregister (engl.: *program counter*, PC). Pro Hardware-Thread werden somit 4,36 KB an Speicher für das Registerfile und ein weiteres Byte für das *Scoreboarding* zur Wahrung der Registerabhängigkeiten benötigt. Hinzu kommen fünf Fehlerbehandlungsregister mit je 32 Bit Breite.

Die Verarbeitungspipeline selbst hat eine Länge von elf Stufen, wobei bereits fünf Stufen den arithmetischen Teil ausmachen (Abbildung 6.6). Um die Zahl der ausstehenden Instruktionen zu vergrößern, erhält jeder Hardware-Thread einen Instruktions-FIFO, welcher bis zu acht dekodierte Instruktionen vorhalten kann. Entgegen der visualisierten Verarbeitungspipeline werden skalare Instruktionen nicht über eine dedizierte ALU verarbeitet, sondern werden parallel auf alle sechzehn Fließkomma-ALUs verteilt, um daraufhin maskiert ausgewertet zu werden. Nicht im Schaubild gezeigt ist die Anbindung des L2-Caches, welcher die beiden L1-Caches mit Daten beliefert. Zu guter Letzt sieht die Architektur eine MMU für die Speicherverwaltung vor. Wird von einer maximal ermittelten ASIC-Taktfrequenz von 570 MHz bei 45 nm Prozesstechnologie ausgegangen, stellt der GPGPU-Rechenbeschleuniger eine maximale Fließkommarechenleistung von 9 GFLOPS je Rechenkern bereit [Bush et al., 2016].

6.2.2 Modellierung des GPU-Simulators

Der innerhalb dieser Arbeit entwickelte Simulator modelliert sowohl den vollständigen Instruktionssatz des GPGPU-Rechenbeschleunigers als auch die Architektur selbst taktgenau. Die einzelnen Komponenten wurden mit der Sprache C++ entwickelt, wobei Klassen einzelne Komponenten der Architektur repräsentieren. Ausgehend von einer Basisklasse wurde die Architektur unterteilt in Rechenkern, Hardware-Thread, Cache – welcher abstrakt als L1- oder L2-Cache genutzt werden kann –, Arbiter, Lade-/Speicherwarteschlangen und Dekoder. Diese Aufteilung wird im Folgenden im Detail beschrieben.

Rechenkern und Verarbeitungspipeline

Der Rechenkern beinhaltet die taktgenaue Verarbeitungspipeline, welche jede Verarbeitungsstufe als Funktion repräsentiert. Um eine schnelle Simulationsgeschwindigkeit zu gewährleisten, wurden ausgewählte Verarbeitungsstufen in Funktionen zusammengefasst und rein durch zeitliche Verzögerungen ersetzt. Beispiele hierfür sind die Fließkomma- sowie Ganzzahlverarbeitungsstufen, welche erst beim Zeitpunkt des Rückschreibens einer Operation tatsächlich ausgewertet werden. Jede Funktion, welche eine jeweilige Verarbeitungsstufe darstellt, ist mit der vorausgehenden sowie nachfolgenden Stufe mit einem zyklischen C++-Ringpuffer statischer Größe verbunden. Dieser Ringpuffer wird zu jedem Zeitschritt mit einem Instruktionspaket von der vorausgehenden Stufe befüllt, wodurch mit jedem Taktschritt ein Instruktionspaket eine jeweili-

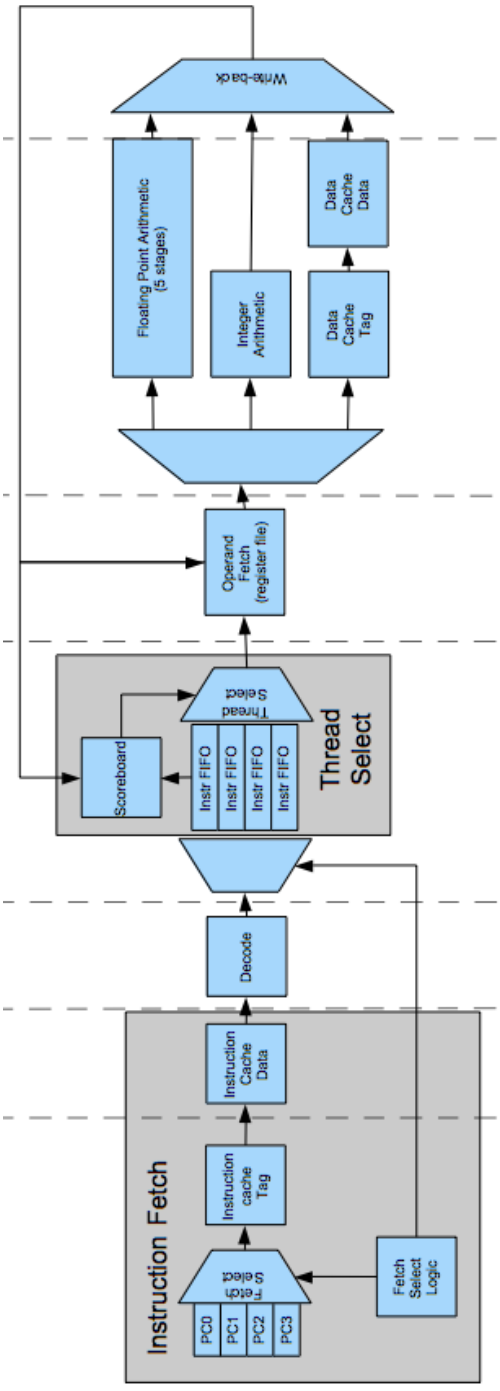


Abb. 6.6: Mikroarchitektur des *NYUZI*-GPGPU-Rechenbeschleunigers [Bush et al., 2016].

ge Verarbeitungsstufe traversiert. Nach einer gewissen Anzahl an Takten kann das Paket von der Verarbeitungsstufe gelesen und mit Hilfe von Rechenarithmetik verarbeitet werden. Im Detail ist dies jedoch nicht der vollständig allokierte Datenblock welcher eine Instruktion repräsentiert, sondern rein ein Zeiger, wodurch ausschließlich Zeiger auf Datenblöcke innerhalb der Verarbeitungsstufen traversieren.

Das jeweilige Instruktionspaket wird in der ersten Verarbeitungsstufe erzeugt, indem ein Pool von vorab allokierten Speicherblöcken angefragt wird. Hierdurch wird dynamische Allokation vermieden. Nach Erhalt wird das Instruktionspaket mit dem aktuellen PC annotiert und traversiert entsprechend mit jedem weiteren Zeitschritt entweder einen Ringpuffer oder eine Verarbeitungsstufe repräsentierende Funktion.

Verarbeitungsstufen, welche in der realen Beschleunigerarchitektur vorhanden, jedoch im Simulator bedingt durch Simulationsgeschwindigkeit eingespart wurden, werden innerhalb des Ringpuffers per weiterem Verzögerungstakt modelliert d.h. der Ringpuffer wird entsprechend um weitere Einträge vergrößert. Obgleich die parallel geschalteten Pipelines für Fließkomma-, Ganzzahl-Alu und Datenlade-/Speichereinheit für den L1d unterschiedliche Latenzen vorsehen, wurden diese zu einer einzelnen Verarbeitungspipeline und somit Ringpuffer zusammengefasst. Sofern eine der drei genannten Pipelines genutzt wird, wird je nach Länge derer das Instruktionspaket in die jeweilig vorgesehene Stelle im Ringpuffer eingefügt. Zugleich wurde diese Pipeline noch um die zwei Zyklen dauernden Operandenladestufen verlängert. Dies geschieht im Einklang mit der Architektur, welche einerseits eine In-Order-Pipeline betreibt und andererseits dafür garantiert, dass nur eine verarbeitete Instruktion pro Takt die Rückschreibstufe erreicht. Entsprechend muss der Simulator ein dekodiertes Instruktionspaket nur in die richtige Verzögerungsstelle des kombinierten Ringpuffers einsetzen, damit das korrekte Zeitverhalten gewährleistet ist.

Der *NYUZI*-GPGPU-Rechenbeschleuniger benutzt zwei getrennte Instruktionsrückabwicklungen (engl.: *rollback*): Sind im Falle eines Sprungs die notwendigen Instruktionen noch nicht im Instruktionscache vorhanden, so wird eine schnelle Rückabwicklung innerhalb der Verarbeitungspipeline getätigt. Da dies früh in der Verarbeitungspipeline geschieht, wird der vordere Bereich von alten Instruktionen befreit und zum fehlgeschlagenen PC zurückgesetzt. Der Hardware-Thread selbst wird zu diesem Zeitpunkt schlafen gelegt und durch den L1-Instruktionscache geweckt sobald die Instruktionen vorliegen. Andererseits existiert eine Maskierung für Instruktionen die bereits vollständig in der Verarbeitungspipeline sind.

Bedingt durch die Einfachheit der GPU-Architektur besitzt diese keine Sprungvorhersage, hätte jedoch auch keine Möglichkeit einen registerinvolvierten Sprung frühzeitig auszuführen, da die Adresse der Sprungstelle erst spät berechnet wird. Auch hier wird der Hardware-Thread durch den L1-Datencache geweckt. Zur Wahrung der Taktgenauigkeit ist die Rückabwicklung von Instruktionen essentiell notwendig, sodass der Simulator dies akkurat modelliert. Hierzu werden Referenzen auf die innerhalb der Verarbeitungspipeline verweilenden Instruktionspakete in dedizierten Listen vorgemerkt, welches zur Folge hat, dass bei einer Rückabwicklung die Liste Verweise auf beeinträchtigte Pakete hat. Zur Beschleunigung der Liste wurde auf die *Intrusive*-Bibliothek in der C++-Bibliothekssammlung *Boost* zurückgegriffen, welche ähnlich zu üblichen C-Verkettungen in die Pakete den Zeiger auf das nächste Element einbettet, wodurch kein C++-STL-Mehraufwand erzeugt wird.

Im Gegensatz zu klassischen Prozessorarchitekturen besitzt die gewählte Architektur bedingt durch die Anlehnung an (GP)GPU-Architekturen für jeden einzelnen Hardware-Thread einen tie-

fen Instruktions-FIFO, welcher standardmäßig acht Einträge tief ist. Dieser wird mit dekodierten Instruktionen befüllt und kann somit ALUs und Lade-/Speichereinheit saturiert halten. Hinzu kommt, dass innerhalb der GPGPU ein sogenannter PLRU-Algorithmus (engl.: *pseudo least recently used*, PLRU) zur zeitlichen Terminierung von ausführbaren Hardware-Threads nutzt. Dies ist nötig, da die Hardware-Threads je nach Ausführbarkeit über die Fließkomma-ALU, die Ganzzahl-ALU und die Datenlade-/Speicherwarteschlange arbitriert werden müssen. Entsprechend existiert innerhalb der Architektur für je einen Hardware-Thread ein FIFO, welcher für den jeweiligen Hardware-Thread dekodierte Instruktionen vorhält. Der Pseudo-LRU-Arbiter kommt hierbei nun zur Anwendung insofern dass er entscheidet, welcher Hardware-Thread im aktuellen Moment auszuführen ist. Da solch Arbiter jedoch rechenaufwändig ist, wurden die dieser vorab berechnet entsprechende als Wertetabellen (engl.: *lookup table*) ersetzt, was sich positiv auf die Simulationsgeschwindigkeit auswirkt.

Dekoder, Registerabhängigkeit und Zwischenspeicher (Caches)

Neben der Verarbeitungspipeline sind die Instruktionsdekodierung sowie die Wahrung von Registerabhängigkeiten wichtige Aspekte bei der genauen Modellierung von Hardware. Auf Basis des innerhalb des *NYUZI*-Emulators vorhandenen Dekoders wurde ein erweiterter und beschleunigter Dekoder, welcher innerhalb von maximal zwei Sprüngen die zur Instruktion passende Arithmetik-, Sprung- oder gar Sonderfunktion dekodiert. Da der Dekoder binärkompatibel zur tatsächlichen RTL-Implementierung entsprechend zur ISA ist, ermöglicht dies die Verwendung des vollständigen Software-Stacks, angefangen von der LLVM-Toolchain inklusive Compiler bis hin zum Betriebssystem.

Zusätzlich zum Dekoder wurden die zur Instruktion passenden (zwei) Registerabhängigkeitsfunktionen implementiert. Die eine kann die Registerabhängigkeitsbits überprüfen, sodass eine Verzögerung ermöglicht werden kann, während die andere die entsprechenden Abhängigkeitsbits je nach Parameter setzen und entfernen kann. Wird eine Instruktionsdekodierung durchgeführt, setzt der Dekoder Verweise (C-Zeiger) auf die drei entsprechenden Instruktionsfunktionen in das innerhalb der Verarbeitungspipeline genutzte Instruktionspaket ein. Erreicht dieses die *Thread Select*-Verarbeitungsstufe (Abbildung 6.6) – die Stufe an welcher der PLRU-Arbiter den Hardware-Thread auswählt – und soll ausgeführt werden, wird zuerst die Funktion zur Überprüfung auf Registerabhängigkeiten ausgeführt. Sind keine Abhängigkeiten vorhanden, werden die entsprechenden Abhängigkeitsbits gesetzt und, sobald die Rückschreibstufe erreicht ist, werden wieder mit der gleichen Funktion die entsprechenden Bits entfernt.

Die zeitliche Terminierung mit Hilfe des PLRU-Algorithmus wird nicht ausschließlich zur Erkennung von ausführbaren Hardware-Threads, sondern zugleich innerhalb der Caches als Austauschstrategie genutzt. Obgleich Caches im Hinblick auf Zeilengröße, Blöcke und Sätze unterschiedlich konfiguriert werden können, funktionieren diese im Grunde immer nach dem gleichen Schema, sodass eine abstrakte Cacheklasse entwickelt wurde. Von dieser leiten der L1i-, L1d- und L2-Cache ab, nutzen jedoch zugleich unterschiedliche Lade-/Speicherwarteschlangen, um Instruktionen sowie Daten ein- und auszuladen.

Flexibilität und Anmerkung

Die im vorigen Abschnitt aufgezeigte Flexibilität des *NYUZI*-GPGPU-Rechenbeschleunigers wurde im Simulator mit Hilfe von `#defines` beibehalten, damit jegliche Eigenschaft des Simulators schnell angepasst werden kann. Unter jegliche fallen die Anzahl an Rechenkernen, Hardware-Threads und Register, sowie die Vektorbreite und der Instruktions-FIFO-Tiefe. Hinzu kommen die Assoziativität, Zeilen- sowie Blockgröße und Austauschstrategie für die jeweiligen Caches. Da all diese Eigenschaften per Präprozessoranweisung definiert werden, erlaubt dies dem Compiler, effizienteren Maschinencode zu generieren, wodurch die Ausführungsgeschwindigkeit beschleunigt werden kann.

NOP-Instruktionen traversieren ebenso die Verarbeitungspipeline und erlauben somit eine taktgenaue Simulation. Dies ist im Grunde nicht zwingend notwendig, jedoch erleichtert dies die Verständlichkeit des Modells und sorgt bei gefüllter Verarbeitungspipeline auch nicht für eine Verlangsamung. Theoretisch wäre es jedoch für weiterführende Studien möglich, einzelne, ausschließlich NOP-Instruktionen beinhaltende Verarbeitungsstufen während eines Taktes zu deaktivieren, wodurch eine leicht erhöhte Ausführungsgeschwindigkeit erzielt werden könnte.

Zusätzlich wurde die Beschränkung seitens der Vektoreinheit gelockert, erzwingt diese beim Laden und Speichern in der RTL-Implementierungen zu Cache-Blöcken "ausgerichtete" (engl.: *aligned*) Adressen. Die Beschränkung kann jedoch, sofern gewünscht aktiviert werden.

6.2.3 Einbettung in SoCRocket

Da nur nur eine sehr begrenzte Auswahl an Werkzeugen zur Modellierung heterogener Systemarchitekturen existiert Abschnitt 5.3, wurde eine Integration in die virtuelle Plattform *SoCRocket* angestrebt [Schuster, 2014]. Hintergrund ist, dass die gegebenen über keine RTL-Implementierung verfügen, wodurch neue Erkenntnisse nicht leicht von der Simulation in reale Hardware überführt werden können. *SoCRocket* bietet hingegen eine Implementierung basierend auf SystemC/TLM-Komponenten in Kombination mit der GRLIB IP-Bibliothek [COBHAM, 1997], wodurch mit Integration des GPU-Simulationsmodells und dessen frei verfügbaren SystemVerilog-RTL-Implementierung eine ganzheitlich frei verfügbare, heterogene Systemarchitektur geschaffen werden konnte [Siegl et al., 2016b]. Durch die simulative Einbettung der Beschleunigerarchitektur verhilft dies *SoCRocket* einerseits zu einer graphischen Ausgabe, sowie andererseits zu einer Beschleunigung von allgemeinen Rechenaufgaben.

Wohingegen die RTL-Implementierung des GPGPU-Rechenbeschleunigers bereits auf den modernen Bus-Standard AXI zurückgreift, setzt die *SoCRocket*-Plattform auf den Vorgänger AMBA. Somit wurde zur Anbindung des GPU-Simulationsmodells ein AMBA-Master/Slave-Schnittstelle modelliert. Diese erlaubt eine simulierte physikalische Kopplung des GPU-Simulationsmodells mit dem AMBA-Bus zwecks Adressierung und Datenaustausch (Abbildung 6.7), wodurch der Grafikkartenspeicher direkt adressiert werden kann. Gleiches gilt für die Register des GPGPU-Rechenbeschleunigers, welche per *Memory-Mapped* IO (MMIO) ebenfalls in den Adressraum des Prozessors abgebildet wurden.

Der neueste Stand der *NYUZI*-RTL-Implementierung setzt zugleich eine MMU ein, demzufolge der Hauptspeicher des Prozessors auch dem Beschleunigers zur Verfügung gestellt werden könnte. Das verwendete Modell besaß zum Zeitpunkt der Implementierung jedoch noch keine

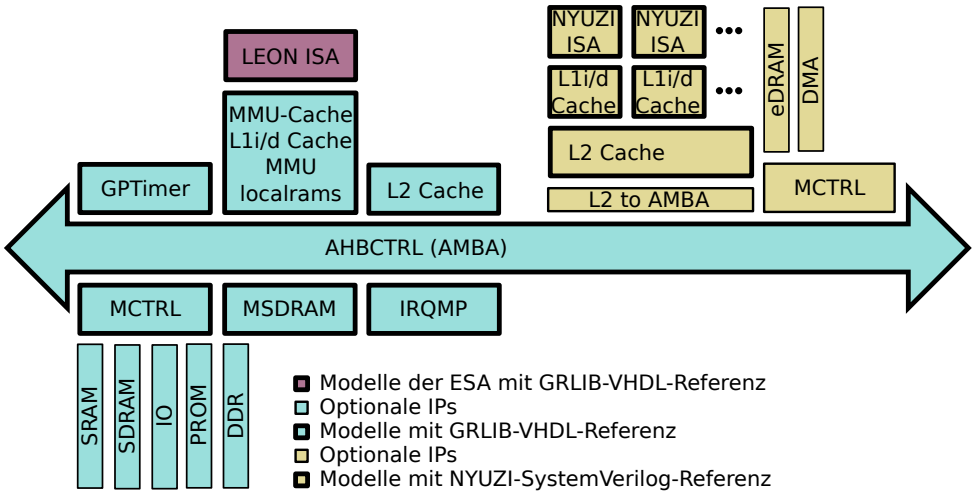


Abb. 6.7: Schaubild der Integration des *NYUZI*-Simulationsmodells in die virtuelle Plattform *SoCRocket*.

MMU und demzufolge konnte dieses Detail noch nicht adressiert werden. Da es jedoch keinen Einfluss auf die Rechenleistung hat, sofern von statisch gepinnten Speicheradressen ausgegangen wird, ist dies für die weitere Analyse kaum von Belang.

Die Konfiguration des GPU-Simulationsmodells sowie die Delegation von allgemeinen Rechenaufgaben an dieses wird vom Prozessor der *SoCRocket*-Plattform initiiert. Dabei lädt diese den Programmquelltext sowie die Daten aus dem Hauptspeicher in den dedizierten Grafikkartenspeicher, woraufhin ein internes Start-Bit innerhalb des Beschleunigers vom Host-Prozessor getriggert wird. Nachfolgend bootet der Beschleuniger und bearbeitet den entsprechenden Programmquelltext. Beim Booten selbst werden die jeweiligen anderen Rechenkerne sowie Hardware-Threads gestartet. Nach dem erstmaligen Start kann eine Delegation über den Grafikkartenspeicher erfolgen, da beide d.h. Prozessor und Beschleuniger direkten Zugriff darauf haben, wodurch ein Austausch stattfinden kann.

6.3 Integration des GPGPU-Rechenbeschleunigers in das HMC-Simulationsmodell

Um eine holistische speichernahe Verarbeitung d.h. einen PIM zu modellieren, wurden die zuvor entwickelten Simulationsmodelle GPGPU-Rechenbeschleuniger und HMC-Speicherarchitektur ineinander integriert. Hierzu wurde auf Basis der AMC-Kenngrößen die Gewichtung zwischen Rechenkern und HMC-*Vault* bemessen, sowie die gegebenen SRAM-Ressourcen für eine mögliche Cachehierarche eines Rechenkerns in Betracht gezogen, sodass eine realistische PIM-Einbettung ermöglicht wurde. Die Schnittstellen zwischen beiden wurden hingegen äußerst flexibel gehalten, wodurch für eine Evaluierung unterschiedliche Faktoren wie Taktfrequenz, Speicherbandbreite und Latenz berücksichtigt werden können. Zusätzlich wurde, inspiriert durch das Höchstleistungsrechnen, eine Hardwarebarriere in das GPU-Simulationsmodell eingeführt, welche für schnellere Synchronisation zwischen den Hardware-Threads sorgen soll. Neben den Hardwareaspekten geht der Abschnitt zugleich auf die Softwareanpassungen ein und zeigt Simulator-Erweiterungen auf, welche einem auf dem Simulator ausgeführten Gastprogramm einen einfa-

chen Datei-Lese- und -Schreibzugriff erlauben.

6.3.1 Schnittstellen und Frequenzdomänen

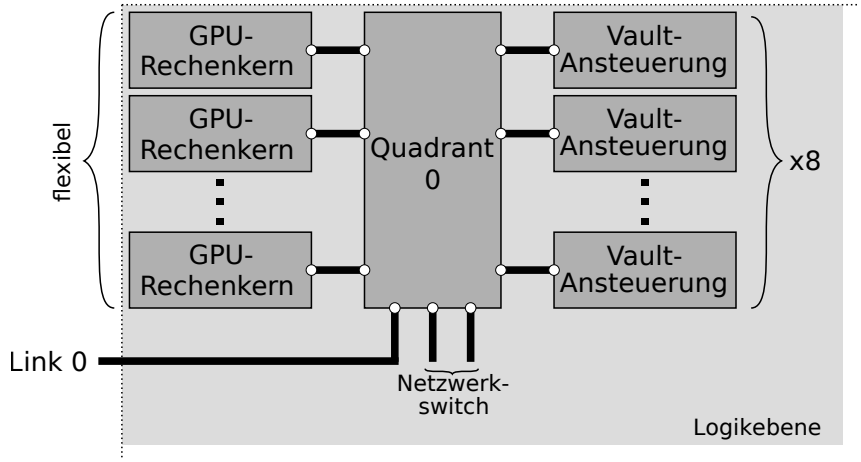


Abb. 6.8: Auszug aus der HMC-Logikebene, bei welcher beispielhaft ein mit PIM-Kapazität in Form von GPU-Rechenkernen ausgestatteter Quadrant dargestellt wird.

Ausgehend vom bekannten Aufbau der HMC-Logikebene (Abbildung 3.6, Abbildung 3.5) wurde ein jeweiliger Quadrant des HMC-Simulationsmodells, um weitere interne Schnittstellen erweitert (Abbildung 6.8). Somit können unterschiedlichste Rechenkerne als eingebettete PIM-Beschleunigerarchitektur evaluiert werden. Angelehnt an die AMC-Studie (Tabelle 5.4), bei welcher acht VLIW-*Lanes* je Quadrant genutzt wurden, sind je Quadrant maximal acht Rechenkerne vorgesehen, da mehr das Eins-zu-Eins-Verhältnis zwischen Rechenkern und *Vault* übersteigen würde. Dies würde bedeuten, dass mehrere Rechenkerne die maximale Speicherbandbreite von $8\,\frac{\text{Byte}}{\text{Takt}_{\text{HMC}}}$ eines *Vaults* teilen müssten. Somit sind Konfigurationen von 4, 8, 16 und 32 GPU-Rechenkernen je HMC realisierbar.

Im Detail stellen die hinzugekommenen Schnittstellen “interne” SLID dar. Dadurch mussten

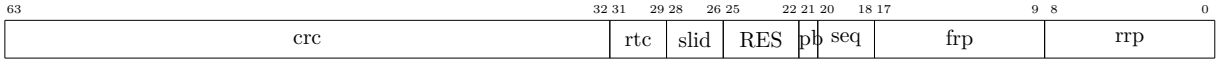


Abb. 6.9: Header-Aufbau des Anfragepakets.

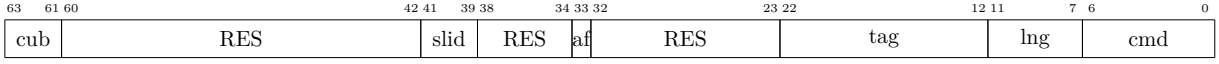


Abb. 6.10: Header-Aufbau des Antwortpakets.

im Bezug auf die HMC-Spezifikation rein der jeweilige Paket-Header d.h. die Feldgröße für die SLID angepasst werden, welcher für nachfolgende Experimente ausreichend Reservebits (siehe *RES*) zur Verfügung stellt (Abbildung 6.9, Abbildung 6.10). Zugleich wurde jede Quadranten-Logik und somit die Endpunkte des internen Netzwerks um die entsprechende Dekodierung erweitert.

Nicht nur die Anzahl an Rechenkernen, sondern auch die Taktfrequenz, mit welcher Rechenkern und HMC betrieben werden sowie die Bitbreite des Busses zwischen HMC und PIM-

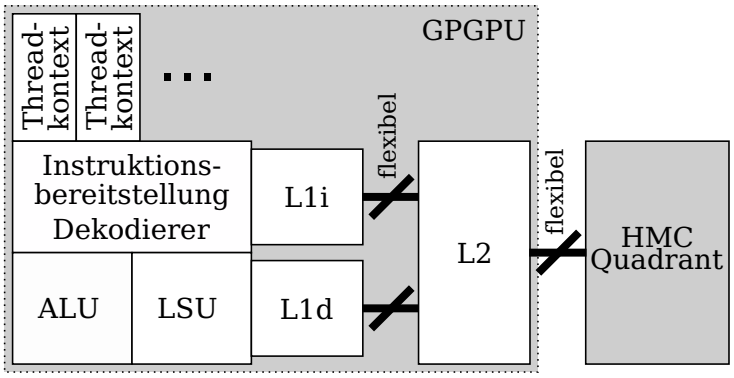


Abb. 6.11: Schemata der Speicherhierarchie zwischen GPU-Rechenkern und HMC-Quadranten.

Rechenkern, wurde flexibel gehalten (Abbildung 6.11). Zugleich kann die Bitrate zwischen Rechenkern und Quadrant individuell eingestellt werden.

Wohingegen die maximale Taktfrequenz der RTL-Implementierung bei einer 45 nm-Technologie 570 MHz beträgt, arbeitet HMC mit einer Taktrate von 1,25 GHz, sodass die Schnittstelle eine Taktanpassung tätigt (Tabelle 5.4). Trotz nur 570 MHz und ausgehend von

HMCs	Rechenkerne	Frequenz		
		570 MHz	625 MHz	1,25 GHz
1	1	9,12 GFLOPS	10 GFLOPS	20 GFLOPS
	4	36,48 GFLOPS	40 GFLOPS	80 GFLOPS
	8	72,96 GFLOPS	80 GFLOPS	160 GFLOPS
	32	291,84 GFLOPS	320 GFLOPS	640 GFLOPS
8	32	2,34 TFLOPS	2,56 TFLOPS	5,12 TFLOPS

Tab. 6.2: Erzielbare Fließkommarechenleistung im Zusammenhang mit der Taktfrequenz, sowie der Anzahl an HMC und Rechenkernen.

einer unmodifizierten Rechnerarchitektur d.h. 16 $\frac{\text{FLOP}}{\text{Takt}_{\text{GPGPU}}}$ und vier Hardware-Threads je Rechenkern, und angenommen 32 Rechenkerne je HMC vermag diese Konfiguration bereits knapp 292 GFLOPS bei einfacher Fließkommagenauigkeit zu erzielen (Tabelle 6.2). Werden darüber hinaus die maximal acht HMCs pro zentralem Prozessor verschaltet, so sind bereits 2,34 TFLOPS bei 1024 Hardware-Threads ($= 32 \cdot 4 \cdot 8$) und 45 nm-Prozesstechnologie möglich. Da die gewählte 45 nm-Prozesstechnologie auf einer Open-Source-Universitätsbibliothek aufbaut, kann angenommen werden, dass mit einer Industrievariante höhere Frequenzen erzielbar sind. Wird davon ausgegangen, dass zumindest mit der Hälfte oder gar mit der vollen Taktfrequenz des HMC betrieben werden kann, so vermag der GPGPU-Rechenbeschleuniger bereits respektable 2,56 TFLOPS bzw. 5,12 TFLOPS zu erzielen[1]. Nicht weiter ausgeführt, jedoch mit dem erbrachten Simulator möglich, sind zukünftige Optimierungen, um die Rechenleistung mit Hilfe von architektonischen Anpassungen oder mit einer höheren Taktfrequenz zu verbessern.

[1]Zum Vergleich: Die Grafikkarte des Typs Tesla P100 der Firma Nvidia erzielt mit 16 nm-Prozesstechnologie, 3584 CUDA-Rechenkernen sowie einer Frequenz von 1,12 GHz 10,6 TFLOPS bei einfacher Fließkommagenauigkeit [Nvidia, 2016].

6.3.2 SRAM- und SLID-Ressourcenoptimierung

Die *NYUZI*-GPU verwendet eine *Harvard-Architektur* bei welcher die L1i- und L1d-Caches jeweils eine Kapazität von 16 KB (Blöcke: 4, Sätze: 64) besitzen (Abbildung 6.11). Hinzu kommt ein L2-Cache mit einem Umfang von 128 KB (Blöcke: 8, Sätze: 256). Zum Verdrängen der Cache-Zeilen wird auf den PLRU-Algorithmus zurückgegriffen, wobei die Blockgröße auf beiden Ebenen 64 Byte beträgt. Auf Ebene eins des Cache wird *Write-Through*-Strategie genutzt, wohingegen auf Ebene zwei *Write-Back* verwendet wird. Zugleich benötigt jeder Hardware-Thread eines jeweiligen Rechenkerns 2720 Byte an SRAM (Tabelle 5.4).

Wiederum angelehnt an einen AMC werden die maximal nutzbaren Ressourcen bemessen (Tabelle 5.4), wobei eine einzelne AMC-VLIW-*Lane* insgesamt 25 704 Byte an SRAM veranschlagt. Soll die Anzahl an Registern pro Hardware-Thread der GPGPU-Rechenbeschleuniger beibehalten werden, bedeutet diese geringe Verfügbarkeit von SRAM einen signifikanten Einschnitt in der Cachehierarchie. Jedoch bekräftigt AMCs sogenannte *lib* und dessen nicht vorhandene Cachehierarchie, dass ausschließlich kleine, innere und somit kritische Schleifen innerhalb des PIM-Rechenbeschleunigers vorgehalten werden.

Daher kann bereits eine grobe PIM-Ausgangsarchitektur definiert werden, welche weiter in Richtung maximaler Fließkommarechenleistung und damit Recheneffizienz optimiert werden kann. Hierzu sollen vorab die SRAM-Kapazitäten betrachtet werden. Ausgehend von kleinen inneren Schleifen, soll für den L1i-Cache eine 1 KB SRAM-Kapazität (Blöcke: 4, Sätze: 2, Blockgröße: 64 Byte) angesetzt werden, stellt dies doch bereits insgesamt 256 Instruktionen dar. Wird von maximal 17,5 KB SRAM je Rechenkern ausgegangen, so verbleiben 16,5 KB.

Trotz klein gewähltem L1i-Cache, widerfährt dem L2-Cache ein signifikanter Einschnitt, indem dieser auf 8 KB festgesetzt wird. Dessen Konfiguration hinsichtlich der Anzahl der Sätze und Blöcke soll später evaluiert werden. Die Blockgröße hingegen wird von ehemals 64 Byte auf 256 Byte vergrößert, da ein HMC-Speicher mit 256-Byte-Zugriffen die maximale Netto-Speicherbandbreite erzielen kann Auch kleinere Zugriffe vermögen die maximale Speicherbandbreite eines HMCs zu erzielen, jedoch wirken sich dabei die ebenfalls zu übertragenden Meta-Informationen (*Header*, *Tail*) prozentual stärker auf die Netto-Bandbreite aus. Ob diese Festlegung der Blockgröße des L2-Caches tatsächlich erfolgreich ist, soll später ebenfalls evaluiert werden. Zur Beschleunigung wird dem L2-Cache eine “Schreibzusammenfassung” (engl.: *write coalescing*)-Logik spendiert, sodass mehrere Schreibvorgängen auf die gleiche Cachezeile zusammengefasst werden können.

Angelehnt an den AMC wird zur Leistungssteigerung ein jeweiliger L1d-Cache als tiefe Lade-/Speicherwarteschlange ausgeführt. Entsprechend soll für jeden Hardware-Thread zumindest eine Lade-/Speicherinstruktion ausstehend (engl.: *in-flight*) gehalten werden, wodurch mindestens 64 Byte, also eine Vektorbreite, für einen jeweiligen Hardware-Thread zu veranschlagen sind. Dies bedeutet, dass das eigentliche Zwischenspeichern der Daten daraufhin nur noch im L2-Cache tatsächlich zum Tragen kommt, da jede eingeladene Vektorladeinstruktion eine neue Pufferzeile einlädt und die alte verdrängt. Hintergrund ist, dass die Architektur näher an den Speicherblöcken ist, sodass Daten sich ohnehin schneller in dem L2-Cache einfinden werden. Entsprechend sollte sich die Reduktion des L1d-Caches innerhalb der Cachehierarchie kaum bemerkbar machen.

Für die optimierte PIM-Architektur soll auch die Anzahl an Rechenkernen sowie Hardware-

Threads variiert werden. Jedoch stellt das Ziel ein Eins-Zu-Eins-Verhältnis zwischen *Vault* und Rechenkern dar, sodass insgesamt 32 Rechenkerne angestrebt werden. Werden nun für den L1i-

Rechen-	je Rechenkern					Summe	∼AMC
kern(e)	Threads	Register [KB]	L1i [KB]	Warteschlange [B]	L2 [KB]	[KB]	*Lanes*
4	4	10,6	1	256	8	80	2,3
4	8	21,3	1	512	8	123	3,5
4	16	42,5	1	1.024	8	210	6,0
4	32	85,0	1	2.048	8	384	10,9
8	4	10,6	1	256	8	159	4,5
8	8	21,3	1	512	8	246	7,0
8	16	42,5	1	1.024	8	420	11,9
8	32	85,0	1	2.048	8	768	21,7
16	4	10,6	1	256	8	318	9,0
16	8	21,3	1	512	8	492	13,9
16	16	42,5	1	1.024	8	840	23,7
16	*32*	*85,0*	*1*	*2.048*	*8*	*1.536*	*43,3*
32	4	10,6	1	256	8	636	18,0
32	8	21,3	1	512	8	984	27,8
32	*16*	*42,5*	*1*	*1.024*	*8*	*1.680*	*47,4*
32	*32*	*85,0*	*1*	*2.048*	*8*	*3.072*	*86,6*
16	16	42,5	4	1.024	16	1.016	28,7
32	8	21,3	2	512	16	1.272	35,9

Tab. 6.3: Unterschiedliche PIM-Konfigurationen, bei welchen GPU-Rechenkerne, Hardware-Threads sowie die Cachehierarchie auf Basis von SRAM in Relation zu VLIW-*Lanes* des AMC gesetzt wird.

Cache 1 KB, für den L2-Cache 8 KB angenommen und wird von einer variablen Tiefe der Instruktionswarteschlange ausgegangen, so lässt sich eine Tabelle von potentiell möglichen Konfigurationen erstellen, welche jeweils nicht mehr Ressourcen als ein AMC mit 32 VLIW-*Lanes* benötigt (Tabelle 6.3). Es zeigt sich, dass Konfigurationen mit 32 Rechenkernen und 16 sowie 32 Hardware – Threads die verfügbare Menge an SRAM übersteigen. Auch eine Konfiguration mit 16 Rechenkernen und 32 Hardware – Threads kann auf Basis des gesetzten SRAM-Kontingents nichts realisiert werden. Als Alternativen ist eine etwas kleinere Konfigurationen mit entweder weniger Rechenkernen sowie eine weitere mit weniger Hardware-Threads angegeben, bei welcher dafür jedoch der L1i- und L2-Cache mit größerer Kapazität betrieben werden können.

Für die weiteren Experimente soll auf einen HMC mit Ringbus zurückgegriffen werden. Wohingegen ein HMC eine maximale, externe Schnittstellenbandbreite von 480 GB/s zur Verfügung stellt, verteilt sich diese im Detail auf vier SLIDs welche jeweils 120 GB/s aggregiert und so 60 GB/s je Richtung zur Verfügung stellt. Entsprechend kann eine Konfiguration mit einem GPU-Rechenkern nur 120 GB/s der internen, ebenfalls aggregierten 320 GB/s DRAM-Speicherbandbreite abrufen. Es ist zu beachten, dass bedingt durch die interne Vernetzung innerhalb des HMCs die vollständige Bandbreite in Theorie jedem GPU-Rechenkern zur Verfügung steht, jedoch die externe Schnittstelle dies limitiert. Somit müssen mehrere GPU-Rechenkerne betrieben werden, um die vollständige DRAM-Speicherbandbreite zu konsumieren (Tabelle 6.4).

Rechen-kern(e)	SLID-Bandbreite je Rechenkern [GB/s]		DRAM-Bandbreite je Rechenkern [GB/s]		SLID Anzahl (genutzt)	*Anmerkung*
	aggregiert	einfach	aggregiert	einfach		
1	120	60	320	160	4 (1)	*Standardkonform*
4	120	60	80	40	4 (4)	*Standardkonform*
8	60	30	40	20	8 (8)	
16	30	15	20	10	16 (16)	
32	15	7.5	10	5	32 (32)	

Tab. 6.4: Gleichmäßige Aufteilung der max. aggregierten SLID-Bandbreite von 480 GB/s auf unterschiedliche GPU-Rechenkern Konfigurationen bei einem 1:1 Verhältnis von GPU-Rechenkern zu SLID. Obgleich die aggregierte DRAM-Speicherbandbreite von 320 GB/s jedem Rechenkern zur Verfügung steht, kann bei parallelem Betrieb aller GPU-Rechenkerne eine theoretische Verteilung ausgemacht werden.

Es zeigt sich, dass eine standardkonforme Konfiguration bestehend aus vier GPU-Rechenkernen bereits die aggregierte Speicherbandbreite saturieren kann. Nicht im Standard vorgesehen, jedoch für die nachfolgende Evaluation essentiell, sind Konfigurationen mit 8, 16 und 32 GPU-Rechenkernen. Entsprechend ist jeweils eine Halbierung der zur Verfügung stehenden Schnittstellenbandbreite vorgesehen. Wie bereits erwähnt sinkt auch bei parallelem Abgriff der Speicherbandbreite der Anteil dessen was ein Rechenkern nutzen kann, obgleich die vollständige Bandbreite bedingt durch den internen Ringbus jedem Teilnehmer zur Verfügung steht.

Zuletzt und da für die PIM-Ausgangsarchitektur nicht in Betracht gezogen, wird die optimierte Architektur jeweils mit 625 MHz und 1,25 GHz betrieben, um zu erkennen ob die gegebene Speicherbandbreite eher durch höherer Taktfrequenz oder durch weitere Rechenkernen und mehr Hardware-Threads saturiert werden muss, um maximale Rechenleistung zu erzielen. Offensichtlich ist, dass weitere Rechenkerne gleichzeitig mehr Leistung zur Verfügung stellen; andererseits jedoch die Effizienz zu evaluieren ist, d.h. ob diese theoretische Leistung auch tatsächlich genutzt wird.

6.3.3 Dedizierte SPMD-Hardwarebarriere

In der Literatur existiert eine Vielzahl unterschiedlicher Implementierungen von Barrieren (Beispiele: Butterfly, Zentrale Liste, Tournament, BST, CPB), welche eine Gruppe von parallel arbeitenden Threads zu einem definierten Zeitpunkt synchronisieren soll. Im Detail bedeutet dies, dass jeder partizipierende Thread an einem vorab definierten Zeitpunkt auf alle anderen warten muss, um nach Eintreffen des letzten einer solchen Thread-Gruppe entsprechend synchronisiert den weiteren Kontrollfluss wieder aufzunehmen.

Da Software jedoch einen Mehraufwand darstellt und die einzelnen Rechenkerne sowie deren Threads sich alle auf dem gleichen Substrat eines PIMs wiederfinden, bietet sich eine Hardwarelösung zur Beschleunigung an. Angelehnt an den Rechenbeschleuniger des Typs Parallella der Firma Adapteva [Olofsson et al., 2014], welcher eine Hardwarebarriere besitzt, wurde die *NYUZI*-ISA um die Instruktion `spmdbarrier` erweitert. Diese führt intern, sofern durch die Programmcode ausgelöst, zu einer globalen Hardwarebarriere, welche alle Hardware-Threads

umfasst. Da der vorgesehene Algorithmus wie im Höchstleistungsrechnen üblich als SPMD ausgeführt wird, ist dies für das Anwendungsszenario ausreichend, kann jedoch mit Hilfe weniger Parameter um eine Partitionierung auf weniger Rechenkerne oder Hardware-Threads für zukünftige Anwendungen optimiert werden.

Führt ein Hardware-Thread die neu eingeführte Instruktion aus, so setzt dieser ein zu seiner Thread-Identifikationsnummer zugehöriges Bit innerhalb eines Rechenkern-internen Registers, das wiederum als Bitmaske agiert. Daraufhin wird eine Rückabwicklung des Hardware-Threads erzwungen und das Aktivbit des Hardware-Threads deaktiviert. Sobald alle Bits innerhalb der Bitmaske gesetzt sind, signalisiert der GPU-Rechenkern den jeweils anderen vorhandenen Rechenkernen das lokale Eintreffen aller Hardware-Threads. Haben alle Rechenkerne signalisiert, dass ihre lokalen Hardware-Threads die Barriere erreicht haben, so wird die jeweils eigene Bitmaske zurückgesetzt und die Aktivbitmaske wieder gesetzt, woraufhin alle Hardware-Threads die Arbeit wieder aufnehmen können.

Die benötigte Hardwarelogik beschränkt sich hierbei auf ein weiteres Register innerhalb des jeweiligen Rechenkerns, sowie wenige Bitleitungen und eine gewissen Menge an UND-Gattern. Somit hält sich der Hardwareaufwand in Grenzen, wodurch ein Algorithmus sich signifikanten Softwaremehraufwand ersparen kann.

6.3.4 Speicheraufteilung und Bootladeprogramm

Das Ziel, eine signifikante Anzahl an Hardware-Threads auf den Rechenkernen der GPGPU zu betreiben, erforderte eine Modifikation der Speicheraufteilung sowie des Bootladeprogramms. Die Originaleinteilung des virtuellen Speichers sieht fünf Bereiche vor, welche mit der Speicher-

Auflistung 6.4: Auszug aus dem original *NYUZI*-Bootladeprogramm.

```
1   .text
2   .globl _start
3   .align 4
4   .type _start,@function
5
6   # Initialisierung einzelner Stacks
7 _start:
8   getcr s0, 0       # Thread Id
9   shl s0, s0, 14    # 16 KB pro Stack
10  li sp, 0x200000   # Globale Stackadr.
11  sub_i sp, sp, s0  # Lokale Stackadr.
12  # [...]
```

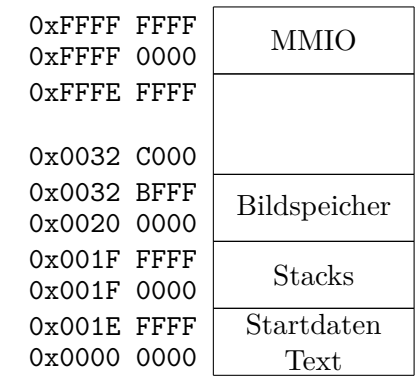


Abb. 6.12: Ursprüngliche *NYUZI*-Speicheraufteilung.

adresse 0x00000000 d.h. dem Programmquelltext sowie globalen Daten beginnt, gefolgt von den jeweiligen Stacks, dem Bildspeicher (engl.: *framebuffer*) und dem dynamisch allokierbaren Bereich (Abbildung 6.12). Der Endbereich stellt die in den virtuellen Adressraum abgebildeten Register (engl.: *MMIO-mapped registers*) dar. Es fällt auf, dass der Bereich des Stacks eine Größe von 64 KB aufweist, wohingegen das Original-Bootladeprogramm je Hardware-Thread einen Stack von 16 KB vorsieht (Aufl. 6.4). Da jeder einzelne Hardware-Thread einen eigenen Stack be-

nötigt, kann die ursprüngliche Speicheraufteilung in Kombination mit dem Bootladeprogramm ausschließlich vier Hardware-Threads bedienen. Da die Allokation absteigend stattfindet d.h. in Richtung der Startadresse, würden weitere Hardware-Threads mit jeweils Anspruch auf einen eigenem Stack den Bereich des Quelltextes und der globalen Daten überschreiten, wodurch ein überschreiben höchst wahrscheinlich wird. Gleichzeitig limitiert dies die Anzahl an Rechenkernen, da nur Konfigurationen von einem Rechenkern mit vier Hardware-Threads, zwei Rechenkernen mit je zwei Hardware-Threads oder vier Rechenkernen mit je einem Hardware-Thread möglich sind.

Obgleich die klassische Speicheraufteilung den Stack am Ende eines Adressraums vorsieht, sodass eine Vielzahl an Stacks unterstützt werden kann, wurde diese Strategie nicht weiter verfolgt. Dies beruht auf der Annahme, dass ausschließlich innere entsprechend kritische Schleifen auf der PIM-Beschleunigerarchitektur genutzt werden (Unterabschnitt 6.3.2). Entsprechend ist davon auszugehen, dass wie im Höchstleistungsrechen üblich, große Speicherallokationen vorab an bestimmte Adressen gepinnt werden. Hinzu kommt, dass ähnlich zu Beschleunigerquelltexten nur wenige Funktionsaufrufe getätigt werden, sodass der Stack auch hierfür nicht oder kaum genutzt wird. Sollten doch Funktionsaufrufe getätigt werden, so können acht verschachtelte gehandhabt werden, indem ein 1 KB großer Stack je Thread vorgesehen ist. Im Detail bedeutet ein Funktionsaufruf, dass der Aufrufer nach vorgegebener Definition dazu gezwungen ist, eine gewisse Teilmenge an Registern zu speichern, wohingegen der Aufgerufene sich um die restlichen Register kümmern muss. Im schlimmsten Fall, d.h. sofern alle Register direkt gespeichert werden müssen, würde sich dies bei der aktuellen 32 Bit-Architektur bei 32 Registern auf maximal 128 Byte an Daten je verschachteltem Funktionsaufruf belaufen. Wird die Architektur für die Zukunft auf 64 Bit angepasst, würde sich diese Größe verdoppeln. Daher wurde der Stack pro Hardware-Thread von ursprünglich 16 KB auf die genannten 1 KB reduziert (Aufl. 6.5). Auch wurde die Basisadresse des Stacks auf 0x300000 angehoben, was zur Folge hat, dass aufgrund des geringen Profils des Stacks bis zu 1044 Hardware-Threads unterstützt werden können (Abbildung 6.13). Dies wurde möglich, da der Bildspeicher für die neue Speicheraufteilung nicht mehr vorgesehen wurde, da dieser ohne Relevanz für die Evaluierung des entworfenen PIM ist. Zusätzlich wurden die MMIO-Register um 0xF000 nach oben verschoben, um für große Experimente Platz zu bieten. Die Größe von Text und Startdaten wurde nicht weiter angepasst, da für eine nachfolgende Evaluierung ein ausreichend großer Bereich für kompilierten Quelltext bereit stehen sollte.

6.3.5 Statusregister für physikalische Identifizierung

Zur globalen Identifizierung eines einzelnen Hardware-Threads bietet die ursprüngliche ISA ein dediziertes Zustandsregister an, welches mit Hilfe von `getcr Register,ID` (engl.: *identifier*, ID) ausgelesen werden kann. Jedoch bietet die Architektur keine Möglichkeit die Anzahl an Rechenkernen und die Menge an Hardware-Threads abzurufen oder gar einen Rechenkern oder lokalen Hardware-Thread zu identifizieren. Daher wurde der Instruktionssatz um weitere dedizierte Spezialregister erweitert, welche die genannten Eigenschaften von der Hardware abrufbar machen. Im Detail lauten die entsprechenden Kodierungen wie folgt:

- `CR_THREAD_ID` (0)
- `CR_CORE_ID` (7)

Auflistung 6.5: Angepasstes Bootladeprogramm, welches bis zu 1088 parallele Hardware-Threads mit jeweils eigenem Stack unterstützt.

```
1   .text
2   .globl _start
3   .align 4
4   .type _start,@function
5
6   # Initialisierung einzelner Stacks
7 _start:
8   getcr s0, 0         # Thread Id
9   getcr s1, 8         # Anzahl Threads
10  getcr s2, 7         # Rechenkern Id
11  mull_i s1, s2, s1
12  add_i s0, s1, s0
13  shl s0, s0,  11     # 1 KB je Stack
14  li sp, 0x300000     # Globale Stackadr.
15  sub_i sp, sp, s0    # Lokale Stackadr.
16  move s0, 0          # Setze argc auf 0
17  call main           # Sprung zur main()
18  spmdbarrier         # Hardwarebarriere
19  move s0, 1
20  getcr s1, 0         # Thread Id
21  shl s0, s0, s1
22  li s1, 0xfffff104 # MMIO Halt-Adresse
23  store_32 s0, (s1) # Stoppe Thread
```

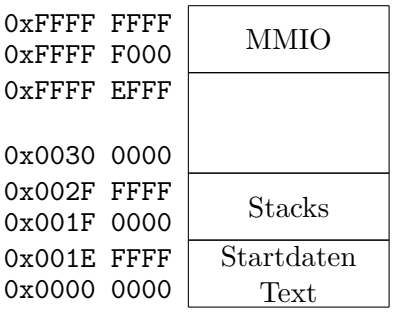


Abb. 6.13: Modifizierte Speicheraufteilung, bei welcher knapp 4 GB an statischen Daten zwischen 0x300000 und 0xFFFFEFFF gespeichert werden können.

- `CR_THREAD_COUNT` (8)
- `CR_CORE_COUNT` (9)

6.3.6 Datei-Lese- und -Schreibzugriff per Statusregister

Die Suche nach Fehlern in einem auf einem Simulator ausgeführten Programmtext gestaltet sich durch den Wegfall von klassischen Terminals sowie Debugger-Schnittstellen als schwierig. Obgleich der Simulator den Speicher lesen und schreiben kann, benötigt es einen Informationsaustausch zwischen der simulierten Anwendung und dem tatsächlichen Anwender. Angelehnt an den Simulator IBM-*mambo*, wurde die *NYUZI*-ISA wiederum erweitert, sodass Teile des Speichers in eine Datei gespeichert oder von einer Datei eingelesen werden können. Die tatsächliche Dateioperation wird dabei vom simulierten Modell an den Simulator delegiert, sodass z.B. Startdaten geladen werden können, ohne die Simulation zu beeinträchtigen.

Im Detail wurde die Instruktion `setcr Wert,ID` mit zusätzlichen, nachfolgend genannten Kodierungen ausgestattet, welche im Detail wären:

- `CR_CALLTHROUGH_ADDR` (13)
- `CR_CALLTHROUGH_FILENAME` (14)
- `CR_CALLTHROUGH_READ` (15)
- `CR_CALLTHROUGH_WRITE` (16)

Da das Setzen eines simulativen Statusregisters genutzt wurde, um die Dateioperationen zu starten, kann jeweils nur ein einzelner Wert an den Simulator übergeben werden. Entsprechend setzt der Programmquelltext vorab erst die für das simulierte Hardwaremodell gesehene Adresse mit Hilfe von `CR_CALLTHROUGH_ADDR`. Ob gelesen oder geschrieben werden soll ist bis zu diesem Zeitpunkt noch unerheblich. Als nächstes wird ein Dateiname mit absoluten Pfad gesetzt (`CR_CALLTHROUGH_FILENAME`). Die Datei muss innerhalb des Dateisystems des Host existieren. Daraufhin wird das Lese- (`CR_CALLTHROUGH_READ`) oder Schreibkommando (`CR_CALLTHROUGH_WRITE`) mit einer vorgegebenen Größe an den Simulator delegiert, welcher wiederum die tatsächliche Dateioperation vollführt.

6.4 Zusammenfassung

Zur Modellierung eines komplexen PIM wurde eine eigenständige, performante Simulationsumgebung entwickelt. Diese beinhaltet ein Simulationsmodell eines dreidimensionalen Speicherstapels sowie ein weiteres für einen Rechenbeschleuniger. Ersteres modelliert einen gemäß der Spezifikation 2.1 standardkonformen *Hybrid Memory Cube*, wobei letzteres eine zur *NYUZI*-ISA binärkompatible GPU implementiert. Die gewählte Beschleunigerarchitektur ist eine Erkenntnis aus Abschnitt 4.5, welches klarstellt, dass ausschließlich angepasste Beschleunigerarchitekturen erhebliche Vorteile bei der Einbettung in einen dreidimensionalen Speicher bieten.

Kernmerkmale des HMC-Simulationsmodells sind dessen Flexibilität, Genauigkeit, Ausführungsgeschwindigkeit und Funktionsumfang. Mit Fokus auf holistische Entwurfsraumexplora-

tionen liegen daher die Komponenten des HMC-Modells, wie etwa Quadrant, *Vault*, *Vault*-Ansteuerung, Arbitrierungslogik, internes Netzwerk und externe Schnittstellen, in einzeln gekapselte Module vor. Entsprechend können diese bei Bedarf ausgetauscht oder mit Hilfe hochgradiger Parametrisierung individualisiert werden. Zusätzlich existiert ein taktgenauer und ein funktionaler Modus, wobei letzterer insbesondere zur schnellen Simulation gedacht ist. Für die Simulationsgenauigkeit wird einerseits auf elementaren Datenpaketen modelliert und andererseits auf den gegen Hardware validierten Simulator *BOBSim* zurückgegriffen. Letzter findet bei der *Vault*-Ansteuerung und zur Formung eines *Vaults* mit mehreren Partitionen sowie Speicherblöcken Anwendung. Im Gegensatz dazu existiert ein funktionales *Vault*-Modell mit fixer Latenz, welches einen äußerst geringen Rechenaufwand besitzt. Zugleich nutzt die entwickelte Logikebene einen speziell konstruierten Taktbaum, welcher es erlaubt nicht verwendete Teile des Simulators dynamisch zu deaktivieren. Anderweitig wäre bedingt durch die Vielzahl an internen Ressourcen eine Ausführung nur äußerst langsam möglich. Im Gegensatz zu Stand der Technik HMC-Simulationsmodellen ermöglicht das entwickelte HMC-Modell die Vernetzung von mehreren HMCs und kann mit dem integrierten Routing-Mechanismus Pakete sowohl innerhalb von HMC sowie extern steuern. Zusätzlich ist jede interne Verbindung mit einer integrierten Aufzeichnungsmöglichkeit verknüpft, sodass traversierende Pakete in anwenderdefinierte Datenbanken oder Dateien geschrieben werden können.

Der GPGPU-Rechenbeschleuniger wurde auf Basis der *NYUZI*-Architektur taktgenau modelliert, welches eine vollständige Implementierung der Verarbeitungspipeline, sowie der einzelnen Caches nach sich zog. Da ein exaktes Modell der Architektur mit Hilfe des Werkzeugs Verilator generiert werden kann, ist neben der Qualität der Beschreibung und der Veränderbarkeit, ein Hauptaugenmerk die Ausführungsgeschwindigkeit. Hierzu traversieren die Daten innerhalb des Simulators nicht über die Cacheebenen, sondern werden direkt vom Hauptspeicher in das jeweilige Register geladen und zurückgeschrieben. Da dies rein in der Simulation geschieht, verändert sich die Laufzeit der auf dem Simulator betriebenen Anwendung nicht, jedoch wird die Simulationsausführung signifikant beschleunigt. Um die Taktgenauigkeit trotz alledem zu garantieren, wird das Verzeichnis eines jeweiligen Cache weiterhin mit entsprechenden Bits simuliert, nur werden die Daten eben nicht zwischengepuffert. Somit können mit dem implementieren Modell deutlich größere Problemstellungen mit entsprechender Taktgenauigkeit betrachtet werden.

Beide Simulationsmodelle, HMC und GPU, wurden jeweils einzeln in unterschiedliche Werkzeuge (SST und *SoCRocket*) eingebettet, wodurch eine Weiterverwendung derer nicht nur als integriertes Modell möglich ist. Jedoch wurde für diese Studie primär die Integration beider vorangetrieben, sodass ein flexibles, holistisches Modellierungswerkzeug für komplexe PIM entstand. Da keine realen in HMC eingebetteten Architekturen bestehen, an welchen sich das Modell hätte orientieren können, und ein realistische PIM-Modellierung erzielt werden sollte, orientieren sich die Hardware-Ressourcen am AMC-Fallbeispiel (Unterabschnitt 4.2.3). Ausgehend von maximal 25 704 Bytes SRAM je AMC-VLIW-*Lane*, reduziert sich somit die Kapazität des L1i-Cache der GPU auf 1 KB und die des L2-Cache auf 8 KB. Die Assoziativität, entsprechend Sätze und Blöcke des jeweiligen Cache, ist flexibel gehalten und soll später evaluiert werden. Der L1d-Cache ist hingegen durch eine tiefere Warteschlange ersetzt, weshalb mehr Lade- und Speicheroperationen zum Kaschieren der Speicherlatenz ausstehen können. Zusätzlich ist eine, angelehnt an Höchstleistungshardware, dedizierte Hardwarebarriere in den GPGPU-Rechenbeschleuniger integriert.

Kapitel 7

Auswertung

An dieser Stelle sollen nun die innerhalb dieser Arbeit entwickelten Simulationsmodelle auf ihre Eignung hin untersucht werden. Zu evaluieren sind zwei Simulationsmodelle, namentlich ein *Hybrid Memory Cube* und eine auf der *NYUZI*-ISA basierende GPU-Architektur, sowie das PIM-Simulationswerkzeug, welches entsprechend eine Integration beider darstellt.

Das vorausgehende Kapitel wies hierzu im Rahmen der HMC-Modellierung auf die Aspekte Flexibilität, Genauigkeit, Ausführungsgeschwindigkeit und Funktionsumfang hin. Um größere Problemstellungen betrachten zu können, stellt dabei der Schwerpunkt die Simulationsgeschwindigkeit dar, weshalb das HMC- und das GPU-Simulationsmodell hierauf explizit evaluiert werden. Ersteres wird anhand von Speicherverlaufsaufzeichnungen aus der SPEC-CPU2006-Benchmarksuite, letzteres mit Hilfe eines portierten Algorithmus analysiert. Der hierbei gewählte Algorithmus ist eine seismische Reverse Time Migration, die mit einem 9-Punkt-*Stencil* betrieben wird. Diese steht repräsentativ für die Klasse an HPC-Anwendungen, welche beschränkt durch und damit sensibel auf Speicherbandbreite sind. Entsprechend wird die RTM auch für das PIM-Simulationswerkzeug verwendet; einerseits zur Demonstration und als Machbarkeitsnachweis, andererseits um die unterschiedlichen Konfigurationen zu evaluieren. Hierbei sollen Rückschlüsse aus dem Beschleunigungseffekt durch speicherintegrierte Verarbeitung gezogen werden, um erzielbare Recheneffizienz sowie PIM-Profitabilität für das Höchstleistungsrechnen zu illustrieren.

Da das HMC-Modell auf einer eigens entwickelten Logikebene und für die *Vaults* auf dem Hardware validierten *BOBSim* Simulator baut, muss nur die Logikebene auf Taktgenauigkeit untersucht werden. Erschwert durch Mangel an realer Hardware und bedingt durch die ungenaue HMC-Spezifikation, wird ein Paket durch den integrierten Ring sowie externen Schnittstellen traversiert und dabei auf zu erwartende Takte untersucht. Im Gegensatz dazu kann das GPU-Simulationsmodell gegen das mit dem Verilator erzeugte Hardware-Modell evaluiert werden. Hinzu kommt eine Untersuchung der Flexibilität einerseits basierend auf der Parametrisierung der Modelle, andererseits im Bereich der Adaption durch nachfolgende Studien, welche unter Umständen nicht nur das integrierte PIM-Simulationswerkzeug, sondern auch einzelne Teile dessen verwenden wollen. Auch wird der Funktionsumfang illustriert wie etwa die Fähigkeit, multiple HMCs miteinander zu vernetzen.

7.1 *Hybrid Memory Cube*-Simulationsmodell

Zur Demonstration des entwickelten HMC-Simulationsmodells wurden aus dem Projekt *Ramulator* die zur Verfügung gestellten Lade-/Speicherverlaufsaufzeichnungen (engl.: *traces*) des bekannten SPEC-CPU2006-Benchmark verwendet (Unterabschnitt 5.1.2). Diese können jeweils als Stellvertreter ihrer Domäne, entsprechend Anwendungsgebiet angesehen werden. Anhand dieser wird die Simulationsgeschwindigkeit mit und ohne optimierten Simulationstaktbaums evaluiert. Zusätzlich wird eine statische Speicherlatenz gegenüber der mit *BOBSim* erzeugten Taktgenauigkeit untersucht, insbesondere im Hinblick auf Simulationsgeschwindigkeit und Mehraufwand durch die gleichzeitige Aufzeichnung mit Hilfe der Datenbank *SQLite*. Da *BOBSim* gegen Hardware validiert ist, stützt sich das entwickelte HMC-Simulationsmodell auf die entsprechende Vorarbeiten, sodass eine tiefe Validierung der einzelnen DRAM-Bänke und -Partitionen nicht mehr notwendig ist. Zu guter Letzt wird eine Multi-HMC-Umgebung illustriert, gerade in Angesicht eines tatsächlich vorhandenen Mechanismus zum Routen elementarer Datenpakete.

7.1.1 Evaluation der Simulationsgeschwindigkeit

Alle vier externen HMC-Schnittstellen wurden an ein C++-Treiberprogramm (engl.: *testbench*) verbunden (Aufl. B.3), welches wiederum den Lade-/Speicherverlauf einer SPEC-CPU2006-Benchmarkanwendung einlas und in entsprechende Lade- sowie Speicheroperationen umwandelte [Henning, 2006, Kim et al., 2016]. Die gegebenen Aufzeichnungen bieten jeweils eine Adresse, eine Verzögerungszeit sowie die Art der Operation, nicht jedoch die Granularität, welche deshalb auf eine in der Rechnerarchitektur üblichen Cachezeilengröße von 64 Byte gesetzt wurde (Aufl. 7.1). Ebenfalls nicht vorhanden ist eine für den HMC wichtige SLID, welche für regulären DRAM aufgrund einer einzigen Schnittstelle nicht definiert ist. Entsprechend werden die Speicherreferenzen gleichmäßig über die verfügbaren SLID-Endpunkte verteilt, wodurch eine jeweils freie Warteschlange innerhalb eines SLID für mehr Geschwindigkeit sorgen kann.

Im Kontrast dazu, dass die hierbei genutzten Lade-/Speicheraufzeichnungen primär zur Demonstration der Simulationsgeschwindigkeit wie auch zur Aufzeichnungsfähigkeit genutzt worden sind, sollten folgende Aspekte nicht unerwähnt bleiben: Eine intelligente Lade-/Speicherwarteschlange könnte jeweils den SLID eines HMC verwenden, welcher den zu adressierenden Quadranten direkt adressiert. Somit müssten keine Quadrantenverbindungen genutzt und entsprechend könnten geringere Latenzen erzielt werden. Darüber hinaus ist anzumerken, dass eine schnell zurückkehrende Ladeoperation in der Theorie dem Prozessor die Möglichkeit eröffnet nachfolgende Instruktion früher zu bearbeiten. Dieses Verhalten kann jedoch mit den gewählten Aufzeichnungen nicht dargestellt werden. Hinzu kommt, dass 64 Byte-Zugriffe die Speicherbandbreite und Bandbreiteneffizienz senken, da ein zu generierendes Netzwerkpaket jeweils eine Anfangssequenz (engl.: *header*) von 8 Byte sowie eine Endsequenz (engl.: *tail*) von 8 Byte besitzt, wobei die maximale und dementsprechend optimale Auslastung 256 Bytes darstellt.

Für die Messungen wurde auf ein Prozessor vom Typ Intel Haswell zurückgegriffen (Tabelle 7.1). Obgleich diese Befehlssatzarchitektur nicht mehr aktuell ist (aktuell gängig ist Intel Kaby Lake, d.h. zwei Generationen jünger), so zeigen die erzielten Simulationszeiten auf, dass eine schnelle bis angemessene Simulation auch mit der rechenintensiven Aktivierung von *BOBSim* realisierbar ist (Tabelle 7.2). Wie eine Vielzahl weiterer Simulatoren und Emulatoren führt das

Auflistung 7.1: Teilaufzeichung von durch einen Prozessor erzeugten Lade-/Speicheranweisungen der Anwendung dealII aus der SPEC-CPU2006-Benchmark wiedergibt. Jede Zeile steht dabei für eine Operation. Zwei Parameter stehen für eine Ladeoperation an, bei welcher die Verzögerung und Adresse angegeben ist. Drei Parameter hingegen stehen für eine Verzögerung, die Ladeadresse und die Rückschreibadresse.

```
...
27 21229632
11 21229760 47764062924992
8 21229696
38 47764062335360
104 47764062335296
119 47764062335552
181 21229504
17 21229440
35 47764062334912
155 47764062335040
110 47764062334976 47764062924800
350 47764062335616
332 47764062335104
355 47764062335488
116 47764062335424
83 21229824
146 47764062335680
185 21229568
659 20241984
25 21230080
11 21230208
8 21230144
58 47764062336384
248 47764062336448
167 21229952
8 21229888
47 47764062335744 47764062925568
...
```

Prozessor	Intel Core i7-4810MQ, 2.8-3,8 GHz
Speicher	2x 8 GB DDR3L-1600
Betriebssystem	Fedora 25
Compiler	g++ 6.2.1-2
Compileroptionen	-Ofast -march=native

Tab. 7.1: Die zur Messung verwendete Testplattform.

Benchmark	Taktzyklen (10^6)	BOBSim	Benachrichtigungen	*SQLite*-Datenbank	Gemessene Zeit (Sek.)	Anfragen pro Sek.	Beschleunigung
403.gcc	203,69				95,53	478	*Fkt. Baseline*
			✓		0,81	56.461	117,91
			✓	✓	79,21	577	1,21
		✓			565,06	81	*Takt. Baseline*
		✓	✓		268,60	170	2,10
		✓	✓	✓	335,36	136	1,68
444.namd	200,00				93,64	229	*Fkt. Baseline*
			✓		0,70	30.612	133,77
			✓	✓	16,66	1.285	5,62
		✓			548,21	39	*Takt. Baseline*
		✓	✓		266,26	80	2,05
		✓	✓	✓	278,24	77	1,97
447.dealII	199,73				93,55	246	*Fkt. Baseline*
			✓		0,69	33.576	135,58
			✓	✓	21,58	1.069	4,34
		✓			534,67	43	*Takt. Baseline*
		✓	✓		272,78	85	1,96
		✓	✓	✓	298,25	77	1,79
458.sjeng	201,89				94,45	762	*Fkt. Baseline*
			✓		0,92	78.520	102,66
			✓	✓	949,11	76	0,10
		✓			555,56	130	*Takt. Baseline*
		✓	✓		282,20	255	1,97
		✓	✓	✓	1064,28	68	0,52
481.wrf	199,82				93,59	292	*Fkt. Baseline*
			✓		0,72	38.039	129,97
			✓	✓	44,34	616	2,11
		✓			540,98	51	*Takt. Baseline*
		✓	✓		270,00	101	2,00
		✓	✓	✓	313,39	87	1,73

Tab. 7.2: Ergebnisse, die im Rahmen der Einspeisung von SPEC-CPU2006-Verlaufsaufzeichnungen von Speicherzugriffen in das erstellte HMC-Simulationsmodell erzeugt wurden. Der HMC wurde dabei mit einer Kapazität von 8 GB, einer Blockgröße von 32 Byte und vier Schnittstellen betrieben. Zugleich war jede Schnittstelle auf maximale Geschwindigkeit, d.h. eine Bitbreite (engl.: *lanes*) von sechzehn und die Bitrate auf 30 GBit/s je *Lane*, eingestellt.

entworfene HMC-Simulationsmodell interne Komponenten sequentiell aus. Somit kann die Simulationsgeschwindigkeit ausschließlich durch eine hohe Taktfrequenz und neue Architekturen des tatsächlich auszuführenden Prozessors beschleunigt werden, nicht jedoch durch das Hinzufügen weiterer Prozessorkerne.

Tabelle 7.2 illustriert, dass das Aktivieren des dedizierten Simulationstaktbaum im funktionalen Modus einen signifikanten Leistungsschub von über einem Faktor 100 erzeugen kann, obgleich die Simulationscharakteristik gleich bleibt. Im taktgenauen Modus hingegen mit insgesamt 32 auszuführenden *BOBSim*-Instanzen je HMC – eine je *Vault*, somit acht je Quadrant – wird noch immer eine Verdopplung der Simulationsgeschwindigkeit ersichtlich. Die Differenz resultiert aus der Rechenintensivität seitens *BOBSim*. Wird die Traversierung der elementaren Datenpakete mit Hilfe der *SQLite*-Datenbank aktiviert, so kann im funktionalen Modus häufig nur noch eine Beschleunigung um den Faktor zwei bis fünf generiert werden, jedoch gibt es mit 458.sjeng einen signifikanten Ausreißer. Weniger stark hiervon betroffen ist der taktgenaue Modus, bei welchem sich die Aktivierung der *SQLite*-Datenbank kaum bemerkbar macht.

7.1.2 (Beschleunigte) statische oder *BOBSim*-getreue Speicherlatenz

Eine Entwurfsraumexploration stellt bedingt durch die große Anzahl an möglichen Kombinationen ein NP-vollständiges Problem dar. Heuristiken und Annahmen werden deshalb genutzt um den aus der NP-Vollständigkeit resultierenden Rechenaufwand zu verkleinern. Meist wird zusätzlich eine schnelle, jedoch unscharfe Breitensuche getätigt. Wurden anhand gewisser Kriterien ideale Kandidaten gefunden, beginnt die Breitensuche bei welcher die Kandidaten weiter eruiert werden.

Wird dies auf das HMC-Simulationsmodell angewandt, so bietet dieses für den ersten Schritt die beschleunigte Ausführung mit Hilfe eines eigens entwickelten Simulationstaktbaums sowie einer statischen Speicherlatenz für die *Vaults*. Werden die interessanten Konfigurationen aus der Entwurfsraumexploration herausgefiltert, so könnte eine vertiefte Analyse mit Hilfe des aktivierten *BOBSim*-Modells getätigt werden, womit nicht nur die Logikebene, sondern auch die *Vaults* taktgenau dargestellt werden. Solch ein Szenario wird hiermit anhand der Anwendung dealII aus der SPEC-CPU2006-Benchmarksammlung beispielhaft und sehr gekürzt aufgezeigt (Abbildung 7.1 und Abbildung 7.2).

Wie beschrieben, wurde das HMC-Simulationsmodell zuerst mit einem deaktivierten *BOBSim* und einer aktivierten *SQLite*-Datenbankanbindung mit Hilfe der dealII Verlaufsaufzeichnung von Lade-/Speicherzugriffen ausgeführt (Abbildung 7.1). Mit den erstellten Messergebnissen kann bereits gezeigt werden, an welchen Stellen innerhalb der internen und externen Verbindungen entsprechende Pakete die meiste Zeit aufgrund von Warten verbrachten. Zugleich wird ersichtlich, wie viele Pakete die jeweilige Schnittstelle zu versenden hatte. Wichtig für den Simulator ist, dass die Menge an Paketen pro Schnittstelle sich nicht verändert, unabhängig davon, ob *BOBSim* aktiviert oder nicht aktiviert wurde. Wird die statische Latenz in Betracht gezogen, so zeigt sich, dass eine starke Auslastung auf den Eingangswarteschlangen herrscht und somit der Durchsatz geschmälert wird. Grund hierfür dürfte für den funktionalen Modus die stetig gleiche Latenz je Paket innerhalb der *Vaults* sein, welche im Kontrast steht zu der nichtlinearen *BOBSim*-Latenz. Somit werden hohe durchschnittliche Wartelatenzen ersichtlich.

Wird *BOBSim* aktiviert, zeigt sich, dass die Anzahl der Pakete je Schnittstelle erwartungsge-

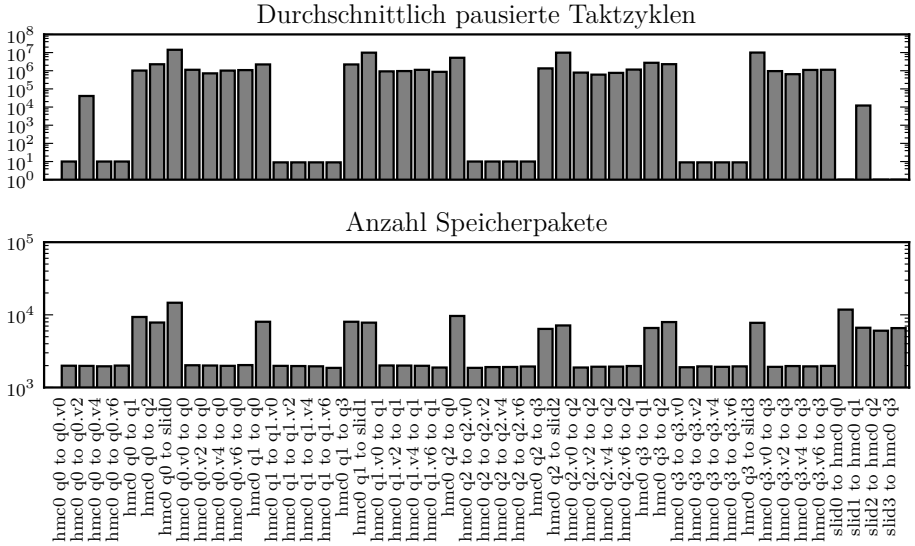


Abb. 7.1: Ergebnisse des HMC-Simulationsmodells, welches mit einer dealII-Lade-/Speicheraufzeichnungen betrieben wurde. Das für Taktgenauigkeit innerhalb der *Vaults* zuständige *BOBSim* ist **deaktiviert**. Auf der x-Achse sind die jeweiligen Verbindungen innerhalb des HMC-Simulationsmodells verkürzt angegeben. D.h. *q* für Quadrant, *v* für *Vault*.

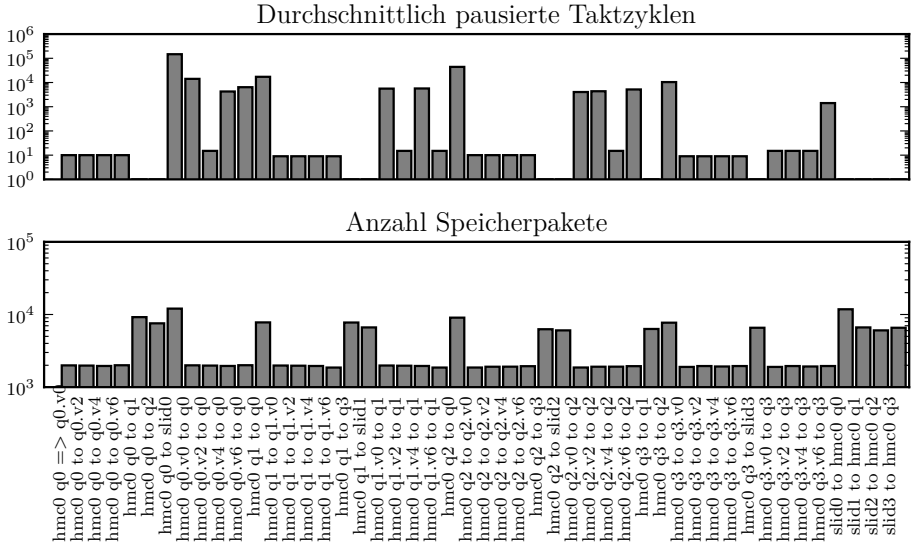


Abb. 7.2: Ergebnisse des HMC-Simulationsmodells, welches mit einer dealII-Lade-/Speicheraufzeichnungen betrieben wurde. Das für Taktgenauigkeit innerhalb der *Vaults* zuständige *BOBSim* ist **aktiviert**. Auf der x-Achse sind die jeweiligen Verbindungen innerhalb des HMC-Simulationsmodells verkürzt angegeben. D.h. *q* für Quadrant, *v* für *Vault*.

mäß gleich bleibt (Abbildung 7.2). Zusätzlich wird ersichtlich, dass nur wenige *Vaults* mit hohen Wartezeiten aufwarten, was sich jedoch zeitgleich auf nahe Warteschlangen ausweitet. Ähnlich wie zur Messung mit der statischen Speicherlatenz weist auch die *BOBSim*-aktivierte Messung eine hohe Wartezeit bei den Endpunkten auf, insbesondere bei Quadrant 0 und SLID-Endpunkt 0. Somit kann der Rückschluss gezogen werden, dass eine schnellere Annahmefunktionalität innerhalb des Treibers notwendig ist.

7.1.3 Multi-HMC-Konfigurationen

Um das Verhalten von multiplen HMCs innerhalb des Simulators zu evaluieren, wurde ein

Auflistung 7.2: Evaluierter *DOT*-Graph, welcher eine Konfiguration mit mehreren HMC-Speichern darstellt. Gegeben sind drei HMC und ein Host, wobei eine Kette aus HMCs gebildet wird. Das Label an jeder Verbindung beschreibt die Verbindungsgeschwindigkeit, welche auf das Maximum, d.h. einer Breite von 16 Bit und Bitrate von 30 GBit/s, gesetzt wurde (HMC-Spezifikation 2.1).

```
graph G {
  rankdir=LR;
  node[style=rounded, shape=record];
  HOST[label="{HOST|{<slid0>slid0|<slid1>slid1|<slid2>slid2|<slid3>slid3
     }}"];
  HMC0[label="{{<link0>link0||<link1>link1}|HMC0|{<link2>link2||<link3>
     link3}}"];
  HMC1[label="{{<link0>link0||<link1>link1}|HMC1|{<link2>link2||<link3>
     link3}}"];
  HMC2[label="{{<link0>link0||<link1>link1}|HMC2|{<link2>link2||<link3>
     link3}}"];

  HOST:slid0 -- HMC0:link0 [label="16,BR30"];
  HMC0:link3 -- HMC1:link0 [label="16,BR30"];
  HMC2:link2 -- HMC1:link1 [label="16,BR30"];
}
```

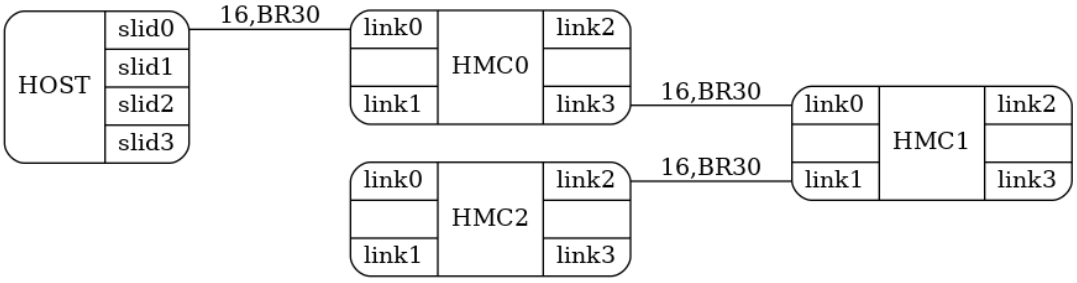


Abb. 7.3: Verbildlichung der evaluierten HMC-Konfiguration aus Aufl. 7.2.

verschlungener, nicht gewöhnlicher Graph mit der Beschreibungssprache *DOT* konstruiert (Auflistung 7.2, Abbildung 7.3). Es ist unwahrscheinlich, dass die gewählte Konfiguration jemals tatsächlich realisiert wird, jedoch bietet sie die Möglichkeit, den Steuerungsmechanismus innerhalb der HMCs zu validieren.

Der Simulator selbst wurde im deaktivierten *BOBSim*-Modus, d.h. hinsichtlich der Speicherbänke im funktionalen Modus betrieben, da rein die Paketdurchquerung eruiert werden sollte. Zur Aufzeichnung und späteren Verfolgung der Paketdurchquerung wurde die Datenbank *SQLite* aktiviert. Wie aus der *DOT*-Konfiguration zu entnehmen ist, sind die jeweiligen Verbindungsschnittstellen auf volle Geschwindigkeit und somit sechzehn *Lanes* bei einer Bitrate von 30 GBit/s gesetzt worden. Bei einer Frequenz von 1,25 GHz entspricht dies einer Übertragungsrate von 48 Bytes pro Taktzyklus (Abschnitt 3.1.2). Zur Vereinfachung sind nicht nur die externen Verbindungsschnittstellen, sondern auch die internen Verbindungsschnittstellen der Quadranten auf die genannten Datenraten und Datenbreiten gesetzt worden. Die TSV-Verbindungen hingegen wurden auf eine Breite von 32 Bits und eine Bitrate von 2,5 GBit/s festgesetzt, womit ein Geschwindigkeit von 64 Bit (8 Byte) pro Taktzyklus ermöglicht wird.

Rein zur Demonstration wird ein einzelnes Paket, welches das Lesekommando *RD256* beinhaltet, in das Netzwerk durch den Host eingespeist. Zu beachten ist, dass ein *RD256*-Lesekommando ein Paket mit einer Größe von 16 Byte (8 Byte Anfangs- und 8 Byte Endsequenz) zur Anfrage erzeugt. Dieses wird wiederum mit einem Paket der Gesamtgröße von 272 Byte (8 Byte Anfangssequenz, 256 Bytes Nutzdaten und 8 Byte Endsequenz) beantwortet. Die Adresse wurde zuvor insofern manipuliert, dass das Paket Daten vom dritten HMC und dessen nullten Quadranten und nullten *Vault* anfordert.

Takt(e)	Rückweg?	Verbindung
1	falsch	slid0 nach hmc0 q0
1	falsch	hmc0 q0 nach q1
1	falsch	hmc0 q1 nach q3
1	falsch	hmc0 q3 nach hmc1 q0
1	falsch	hmc1 q0 nach q1
1	falsch	hmc1 q1 nach hmc2 q2
1	falsch	hmc2 q2 nach q0
2	falsch	hmc2 q0 nach q0.v0
34	wahr	hmc2 q0.v0 nach q0
6	wahr	hmc2 q0 nach q2
6	wahr	hmc2 q2 nach hmc1 q1
6	wahr	hmc1 q1 nach q0
6	wahr	hmc1 q0 nach hmc0 q3
6	wahr	hmc0 q3 nach q2
6	wahr	hmc0 q2 nach q0
6	wahr	hmc0 q0 nach slid0

Tab. 7.3: Aufzeichnung des *RD256*-Lesekommandopakets, welches durch drei HMCs traversierte.

Im Detail benötigt das zum Lesen anfragende Paket mit einer Größe von 16 Byte und einer Übertragungsrate von 48 Byte pro Takt exakt einen Takt je Verbindung (Tabelle 7.3). Einzig die TSV-Verbindung liefert nur 8 Byte pro Takt, wodurch das Paket exakt zwei Takte benötigt, welches ebenfalls in der Aufzeichnung ersichtlich ist. Das gesendete Paket führt im adressierten HMC eine Leseoperation aus, sodass auf dem Rückweg ein Paket mit der Länge von 272 Bytes zurückgesendet wird. Dieses tritt wiederum den Rückweg durch die TSV Verbindung an, wodurch 34 Taktzyklen vergehen, bis das Paket den Quadranten erreicht. Für jegliche weitere Verbindung,

welche nun jeweils 48 Bytes pro Takt Durchsatz bieten, benötigt das Paket sechs Taktzyklen und erreicht damit nach exakt sieben Sprüngen (engl.: *hops*) den anfragenden SLID mit der Nummer 0. Im Kontrast zu anderen Simulatoren modelliert der hierbei entwickelte Simulator auf der FLIT-Ebene, wodurch weitere Pakete den nicht vollständig genutzten sechsten Takt ausnutzen könnten. Hintergrund ist, dass ein FLIT eine Größe von 128 Bit (16 Byte) besitzt, im sechsten Takt für das gewählte Paket ausschließlich 32 Byte der 48 Byte genutzt worden sind. Bedingt durch Subtakte könnte somit noch ein weiteres FLIT von einem weiteren, ebenfalls zu sendenden Paket genutzt werden.

7.2 Fallbeispiel: Zweidimensional vorwärtsgerichtete Reverse Time Migration

Abgeleitet aus der Problemstellung, dass seismische Anwendungen eine äußerst geringe Effizienz bei Fließkommaberechnungen bieten und entsprechend beschränkt durch die Speicherbandbreite sind, wird auf Basis der Vorarbeiten von Medeiros et al. [Medeiros, 2013], Grosser et al. [Grosser et al., 2011] und Siegl et al. [Siegl, 2012, Siegl et al., 2015] das Fallbeispiel einer zweidimensional vorwärtsgerichteten Reverse Time Migration mit 9-Punkt-*Stencil* aufgezeigt. Da diese Variante das Datenformat der Werkzeugsammlung *Seismic Un*x* verwendet, können deren Werkzeuge zur Visualisierung genutzt werden (Unterabschnitt 2.2.7). Im Folgenden soll die gewählte zweidimensional vorwärtsgerichtete Reverse Time Migration mit 9-Punkt-*Stencil* der vierten Ordnung formal notiert werden. Zusätzlich wird ein Einblick in deren Optimierungspotential gegeben, sodass mit der daraus resultierenden arithmetischen Intensität Rückschlüsse auf die Beschleunigungsmöglichkeiten durch zukünftige Beschleuniger und insbesondere PIM-Architekturen gezogen werden können.

7.2.1 Algorithmus

Zur Modellierung eines seismischen Algorithmus wurde eine zweidimensional vorwärtsgerichtete Reverse Time Migration erstellt, welche ohne spezifische Eingabedaten als synthetischer Benchmark Verwendung finden kann. Die gewählte RTM selbst nutzt eine akustisch homogene Wellengleichung, die bedingt durch den Laplace-Operator eine zweite Ableitung aufweist und als erste Ordnung definiert ist (Unterabschnitt 2.2.4). Je nach Anforderungen, etwa Genauigkeit (Approximation der FDM) oder Geschwindigkeit, kann die akustisch homogene Wellengleichung innerhalb einer RTM mit unterschiedlichen *Stencils* (Laplace-Operatoren) verwendet werden. Aufgrund der existierenden Vorarbeiten wurde hierzu ein 9-Punkt-*Stencil* der vierten Ordnung gewählt (Abbildung 7.4).

Ausgehend von einer n-dimensionalen Reverse Time Migration, mit eingesetztem *Stencil* ergibt sich somit (Herleitung: Abschnitt A.2):

$$p(x_1, \ldots, x_n, t + \Delta t) = 2p(x_1, \ldots, x_n, t) - p(x_1, \ldots, x_n, t - \Delta t) + c^2 \Delta t^2 \Delta p$$

Wobei für den zweidimensionalen Fall mit x und z Parametern gilt:

$$p(x, z, t + \Delta t) = 2p(x, z, t) - p(x, z, t - \Delta t) + c^2 \Delta t^2 \Delta p$$

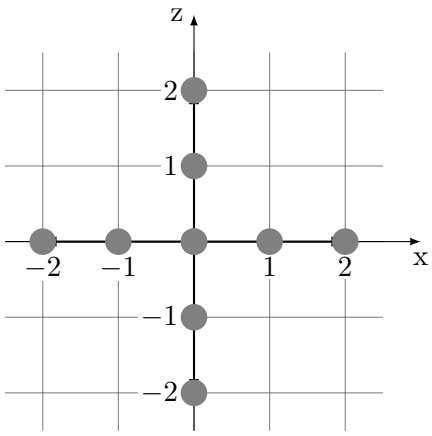


Abb. 7.4: Gewählter 9-Punkt-*Stencil* der vierten Ordnung.

Der Anhand des Laplace-Operators Δp entwickelte zweidimensionale 9-Punkt-*Stencil* der vierten Ordnung (Herleitung: Abschnitt B.1) lautet wie folgt:

$$\Delta p := p_{xx} + p_{zz}$$

$$\Delta p = \frac{1}{12h^2} \begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 16 & 0 & 0 \\ -1 & 16 & -60 & 16 & -1 \\ 0 & 0 & 16 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix} p$$

Es resultiert für die zweidimensional vorwärtsgerichtete Reverse Time Migration mit 9-Punkt-*Stencil* der vierten Ordnung folgende Genauigkeit:

$$p(x,z,t+\Delta t) = 2p(x,z,t) - p(x,z,t-\Delta t) + \frac{c^2\Delta t^2}{12h^2} \begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 16 & 0 & 0 \\ -1 & 16 & -60 & 16 & -1 \\ 0 & 0 & 16 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix} p$$

Aufgrund der aus der Approximation hervorgehenden Instabilität der akustisch homogenen Wellengleichungen müssen – wie bereits in Unterabschnitt 2.2.4 aufgezeigt – folgende Bedingung, die sogenannte CFL-Zahl, eingehalten werden:

$$c = \Delta t \sum_{i=1}^{n} \frac{u_{h_i}}{\Delta h_i} \leq 0,606$$

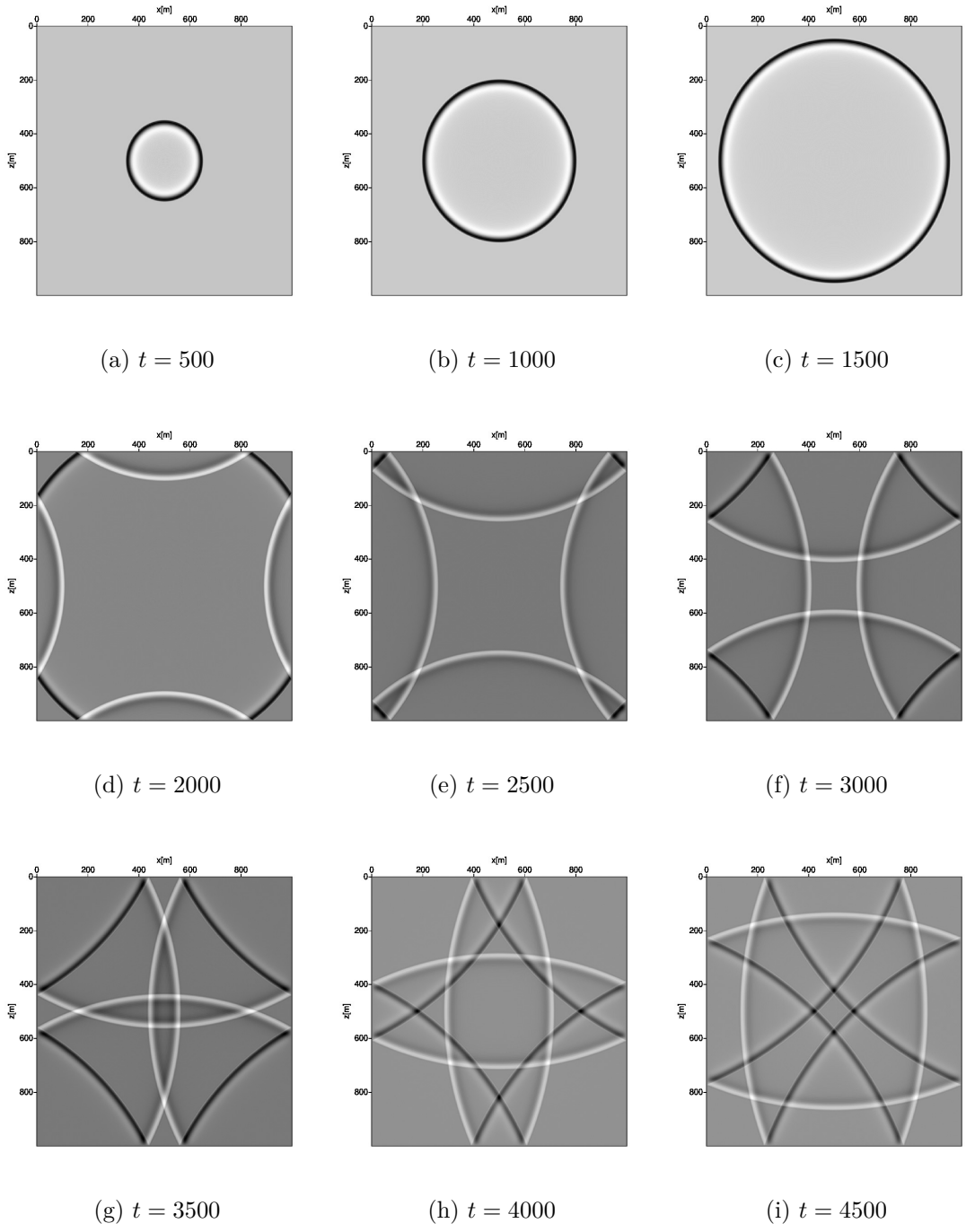
$$\Delta h < \frac{c_{min}}{5 \cdot f_{max}}$$

Für den maximalen Zeitschritt Δt gilt dadurch:

$$\Delta t \leq \frac{0,606 u_h}{c_{max}}$$

Zusätzlich ist die maximale Frequenz des gewählten *Ricker Wavelet* (Unterabschnitt 2.2.6) limitiert durch:

$$f_{max} < \Delta h \cdot c_{min} \cdot 5$$



(a) $t = 500$ (b) $t = 1000$ (c) $t = 1500$

(d) $t = 2000$ (e) $t = 2500$ (f) $t = 3000$

(g) $t = 3500$ (h) $t = 4000$ (i) $t = 4500$

Abb. 7.5: Beispiel einer vorwärtsgerichteten, zweidimensionalen Reverse Time Migration mit 9-Punkt-*Stencil* zweiter Ableitung, vierter Ordnung. Gewählte Parameter hierfür sind $dim_x = dim_z = 1000$, $c_{max} = 1500$, $c_{min} = 0.005$, $h = 4$. (Bilder wurden mit Hilfe der Werkzeuge *Ximage* und *psimage* aus der Werkzeugsammlung CWP *Seismic Un*x* erstellt [Stockwell, 2011].)

Für die Granularität der hergeleiteten Wellengleichung wird in der seismisch-wissenschaftlichen Gemeinschaft zumeist auf Fließkommazahlen der einfachen Genauigkeit (engl.: *single precision*) zurückgegriffen. Wird der erstellte RTM-Algorithmus dabei in eine Implementierung überführt, so können die berechneten Fließkommazahlen mithilfe des innerhalb der *Seismic Un*x* Werkzeugsammlung vorhandenen Werkzeuges *Ximage* sowie *psimage* als seismische Bilder visualisiert werden (Unterabschnitt 2.2.7, Abbildung 7.5).

7.2.2 Beschleunigung

Für jeden Zeitschritt als auch für jeden an der Stelle (x, z) zu berechnenden *Stencil* muss folgende Gleichung berechnet werden:

$$p(x,z,t+\Delta t) = 2p(x,z,t) - p(x,z,t-\Delta t) + \frac{c^2\Delta t^2}{12h^2} \begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 16 & 0 & 0 \\ -1 & 16 & -60 & 16 & -1 \\ 0 & 0 & 16 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix} p$$

Es kann angenommen werden, dass die maximale Anzahl an Zeitschritten, die CFL-Zahl und die *Stencil*-Weite für alle *Stencils* sowie Zeitschritte konstant sind. Entsprechend kann der Term

Auflistung 7.3: Pseudo-Code der seismischen Reverse Time Migration, welcher einen 9-Punkt-*Stencil* mit zweiter Ableitung, vierter Ordnung nutzt.

```
1  // Iteration Zeitschritte
2  for( t = 0; t < TimeSteps; t++ ){
3    // Injektion des Ricker Wavelet
4    // in das aktuelle Druck-Feld
5    APF[x_pulse][z_pulse] += SeismicPulse[t];
6
7    // Iteration X und Z Achsen
8    for( x = 2; x < (dimX-2); x++ ){
9      for( z = 2; z < (dimZ-2); z++ ){
10       // Berechnung des 2-dim. Stencil
11       // Speicherung dessen in folgendes Druck-Array
12       NPF[x][z] = 2.0f * APF[x][z] - PPF[x][z] + VEL[x][z]
13                 * (-60.0f * APF[x][z]
14                    +16.0f * ( APF[x][z-1] + APF[x][z+1]
15                             + APF[x-1][z] + APF[x+1][z] )
16                    -        ( APF[x][z-2] + APF[x][z+2]
17                             + APF[x-2][z] + APF[x+2][z] ) );
18     }
19   }
20   // Vertauschen der Array-Pointer.
21   TMP = PPF; PPF = APF; APF = NPF; NPF = TMP;
22 }
```

$\frac{c^2\Delta t^2}{12h^2}$ einmalig vorab berechnet und als Matrix unter dem Titel *Velocity* (VEL) vorgehalten werden (Aufl. 7.3). Wird ein tatsächlicher geologischer Querschnitt für die Schallgeschwindigkeits-

informationen herangezogen, so kann der konstante Term vorab auf diese multipliziert werden, sodass auch hier rein *VEL* als Zusammenfassung genutzt werden kann.

Wie geschildert, ist das *Ricker Wavelet* für jeden Zeitschritt äußerst rechenaufwändig, kann jedoch vorab berechnet und in einer Vektorstruktur vorgehalten werden (Unterabschnitt 2.2.6). Je Zeitschritt muss somit keine Berechnung durchgeführt, sondern rein der nächstfolgende Wert aus dem Vektor gelesen und in das zweidimensionale Feld injiziert werden. Da der Anwender wählen kann, an welcher Stelle diese Injektion vonstatten gehen soll, kann bei einer gethreadeten Implementierung exakt der Thread dies vornehmen, welcher den (x, z) Abschnitt ohnehin berechnet, wodurch keine weitere Synchronisierung notwendig ist. Bedingt durch das einfache Laden des vorab berechneten *Ricker Wavelet* kann dieses durch diese Optimierung bei großen seismischen Berechnungen vernachlässigt werden.

Zu guter Letzt wird die Handhabung der seismischen Felder optimiert. Da die Zwischenergebnisse nicht von Belang sind, können rein drei Felder angelegt werden, über welche je Zeitschritt iteriert wird. Somit muss je Zeitschritt nur der Zeiger zu den Feldern neu gesetzt werden.

Wird das vorhergehende (engl.: *previous pressure field*, PPF), das aktuelle (engl.: *actual pressure field*, APF), und das folgende Druckfeld (engl.: *next pressure field*, NPF) und VEL näher betrachtet, so können PPF und NPF zusammengefasst und somit als ein Feld genutzt werden, da die alten Ergebnisse ohnehin überschrieben werden (Aufl. 7.3). Somit kann weiterer Speicherbedarf verringert und gleichzeitig größere seismische Problemstellungen berechnet werden. Entsprechend sind nur noch drei Felder notwendig, welche eine minimale Speicherbelegung jeweils der Größe $dim_x \cdot dim_z \cdot 4\,\text{Bytes}$ benötigen (4 Bytes aufgrund einfacher Genauigkeit). Für die *NYUZI*-Portierung wurden die Druckfelder vorab statisch ausgerichtet, damit eine dynamische Allokation von Speicher und somit eine dynamische Allokation auf dem *Heap* nicht notwendig wird (Unterabschnitt 6.3.4). Grund hierfür ist, dass jedes Druckfeld in einem jeweilig anderen Quadranten des HMCs gepinnt wird, sodass bei drei nachfolgenden Zugriffen auf APF, NPF und VEL ein jeweils anderer Quadrant getroffen und somit maximale Parallelität innerhalb des Speicherstapels erzeugt wird (Unterabschnitt 7.2.2).

Weitere Rechenleistungsbeschleunigung kann mit dedizierter Hardware erzeugt werden. Gremm et al. zeigt hierzu eine Vektorisierung des *Stencils* auf, bei welcher der *Stencil* in x-Richtung vervierfacht wird [Gremm, 2011]. Entsprechend werden je Zeitschritt vier *Stencil* parallel berechnet. Entgegen der natürlichen x-Iteration mit anschließender z-Iteration kann dies durch z-Iteration vor x-Iteration umgestellt werden. Dadurch lässt sich eine vektorisierte Spalte mit entsprechender z-Schleife ausrollen damit vorab geladene Vektoren für nachfolgende Zeilen genutzt werden können. Da je Berechnung exakt fünf Zeilenvektoren notwendig sind (abgesehen von Verschiebungen in der Mitte und Laden der beiden Reihenvektoren), können somit vier Zeilenvektoren für die kommende Reihenberechnung innerhalb derselben Spalte wiederverwendet werden.

7.2.3 Roofline-Modell

Unoptimiert erzielt die arithmetische Intensität (AI) der Reverse Time Migration mit gewähltem 9-Punkt-*Stencil* zweiter Ableitung, vierter Ordnung 0,2333 $\frac{\text{FLOP}}{\text{Byte}}$ $(= \frac{14\,\text{FLOP}}{4\,\text{Byte} \cdot (11_{ld} + 3_{ld_{konstant}} + 1_{st})})$ (Aufl. 7.3), sodass die Effizienz des Algorithmus bei vielen Rechnerarchitekturen rein durch die Speicherbandbreite beschränkt wird (Beispielhaft: Abbildung 7.6). Optimiert werden können

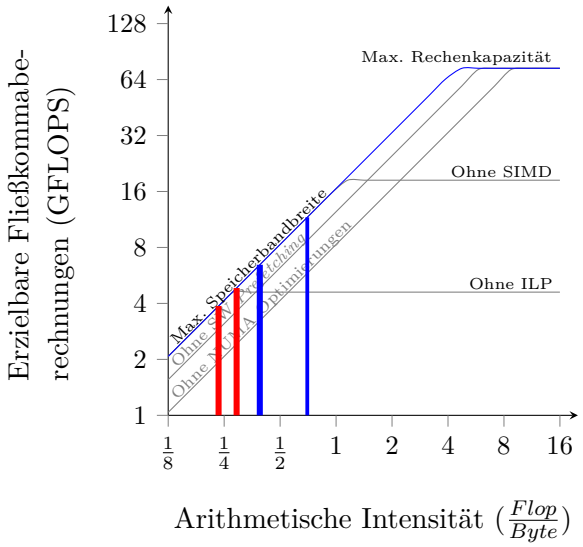


Abb. 7.6: *Roofline*-Modell eines Prozessors vom Typ AMD Opteron 2356 (Barcelona). Vier arithmetische Intensitäten beschreiben verschiedene Optimierungen des 9-Punkt-*Stencils*. Von links nach rechts: Kanonische Implementierung; Vorhalten von Konstanten durch Register; Nutzung von SSE-Vektoren mit einfacher Genauigkeit; Einführung von Vorabladesequenz.

zunächst die drei Konstanten 2, 16 und 60, welche idealerweise in Registern vorgehalten werden. Dadurch verbessert sich die AI auf 0,2917 $\frac{\text{FLOP}}{\text{Byte}}$ ($= \frac{14\,\text{FLOP}}{4\,\text{Byte}\cdot(11_{ld}+1_{st})}$).

Zusätzlich können mit Hilfe von Vektorinstruktionen mehr Daten gleichzeitig vom Speicher geladen werden (Unterabschnitt 7.2.2). Diese müssen zwar bedingt durch die Anordnung des *Stencils* innerhalb der Vektorregister verschoben werden, dies kann jedoch effizient mit Verschiebe-Operationen getätigt werden. Somit kann die gewählte Reverse Time Migration mit 9-Punkt-*Stencils* im Falle der Befehlssatzerweiterung SSE und deren vier parallel zu verarbeitenden Fließkommaoperationen mit einfacher Genauigkeit eine arithmetische Intensität von 0,3888 $\frac{\text{FLOP}}{\text{Byte}}$ ($= \frac{4\cdot 14\,\text{FLOP}}{(4\cdot 7_{ld}+4_{ld}+4\cdot 1_{st})\cdot 4\,\text{Byte}}$) erzielen. Grundsätzliche Idee dabei ist es, die fünf vertikalen Elemente des *Stencils* per Vektorregister zu laden, sodass ein Block mit einer Breite von vier Fließkommawerten und einer Höhe von fünf Fließkommawerten entsteht. Um nun vier *Stencil* parallel zu verarbeiten, benötigt es noch der horizontal Fließkommawerte, welche links und rechts des Mittelpunkts nun entsprechend links sowie rechts des Blockes einzeln geladen werden müssen. Durch Vektorverschiebeoperationen des vertikal gesehenen mittleren Vektors lässt sich nach Laden dieser einzelnen Werte für jeden der – innerhalb des vorab geladenen Blockes – vier *Stencil* die horizontal notwendigen Fließkommawerte zusammensetzen.

Wird von einem typisch sequentiell, zeilenbasiert ladenden Cache-*Prefetcher* abgesehen, kann die innere d.h. spaltenbasierte Schleife ausgerollt werden, sodass vier von fünf Vektorregister vollständig für die kommende Iteration recycelt werden können (Unterabschnitt 7.2.2). Bedingt durch das Laden aller nötigen Vektorregister, erzeugt jede erste *Stencil*-Berechnung einer neuen Spalte eine AI von 0,3888 $\frac{\text{FLOP}}{\text{Byte}}$. Im Gegensatz dazu erzielt jede weitere *Stencil*-Berechnung innerhalb derselbigen Spalte eine AI von 0,7 $\frac{\text{FLOP}}{\text{Byte}}$ ($= \frac{4\cdot 14\,\text{FLOP}}{(4\cdot 3_{ld}+4_{ld}+4\cdot 1_{st})\cdot 4\,\text{Byte}}$. Einzige Bedingung ist, dass Register-Register-Kopierinstruktionen nicht als Ladeoperationen angesehen werden. Dies wird jedoch im klassischen *Roofline*-Modell nicht definiert, da dieses ausschließlich Lade-/Speicheroperationen in Relation zu Fließkommaberechnungen setzt.

In Anbetracht aller beschriebenen Optimierungsgrade zeigt bereits das *Roofline*-Modell am Beispiel eines Prozessors vom Typ AMD Opteron 2356, dass eine Optimierung auf Instruktionsebene nur geringen Leistungsgewinn erzielt (Abbildung 7.6), und auch diesen nur unter Schwierigkeiten. Da jedoch die Reverse Time Migration mit 9-Punkt-*Stencil* äußerst parallel (engl.: *embarrassingly parallel*) betrieben werden kann, können insbesondere GPU-Architekturen mit Hilfe einer massiven Anzahl an speicherlatenzunempfindlichen Hardware-Threads signifikante Beschleunigungseffekte erzielen. Dies zeigte im Jahr 2012 das Team um Grosser et al. auf anhand einer Heterogenisierung mit Hilfe einer CPU/GPU(s)-Kombination [Grosser et al., 2011, Siegl, 2012]. Der erzielte Geschwindigkeitszuwachs variiert je nach Konfiguration, jedoch konnte eine GPU-Beschleunigung um den Faktor 31,2 gegenüber einer reinen Prozessorimplementierung erzielt werden.

Auf diesen Arbeiten aufbauend wurde eine GPU-Portierung auf die *NYUZI*-ISA vorgenommen (Beschleunigermodellierung: Abschnitt 6.2, Implementierung: Aufl. B.1) [Siegl et al., 2016b]. Die so entstandene hochoptimierte RTM-Implementierung erzielt eine maximale AI von 0,8235 $\frac{\text{FLOP}}{\text{Byte}}$ ($= \frac{16 \cdot 14\,\text{FLOP}}{(16 \cdot 3_{ld} + 4_{ld} + 16 \cdot 1_{st}) \cdot 4\,\text{Byte}}$) erzielen (Aufl. B.2). Im Gegensatz zu den ansonsten 0,3888 $\frac{\text{FLOP}}{\text{Byte}}$ für den jeweils ersten *Stencil* je Spalte erzielt die Portierung hingegen eine AI von 0,56 $\frac{\text{FLOP}}{\text{Byte}}$ ($= \frac{16 \cdot 14\,\text{FLOP}}{(16 \cdot 5_{ld} + 4_{ld} + 16 \cdot 1_{st}) \cdot 4\,\text{Byte}}$), wodurch sie trotz Optimierung noch stärker beschränkt durch den Speicher ist.

7.3 GPGPU-Rechenbeschleuniger

Um das Potential des entwickelten Simulators hinsichtlich Taktgenauigkeit, Simulationsgeschwindigkeit sowie Flexibilität aufzuzeigen, werden im Folgenden zwei Anwendungen auf dem Simulator evaluiert. Einerseits wird die Anwendung *Rotozoom* auf dem Simulationsmodell ausgeführt, welcher zeitgleich innerhalb der ganzheitlich, virtuellen Plattform *SoCRocket* als Coprozessor und somit zur Grafikausgabe betrieben wird [Schuster, 2014]. Andererseits wird die in Abschnitt 2.2 hergeleitete seismische Reverse Time Migration mit 9-Punkt-*Stencil* zweiter Ableitung, vierter Ordnung genutzt (Abschnitt B.2). Zur Wahrung der Gerechtigkeit wurden alle

Rechenkerne	1
Hardware-Threads	4
Cache-Zeile	64 Byte
L1i-Sätze	64
L1i-Blöcke	4
L1i-Kapazität	16 KB
L1d-Sätze	64
L1d-Blöcke	4
L1d-Kapazität	16 KB
L2-Sätze	256
L2-Blöcke	8
L2-Kapazität	128 KB

Tab. 7.4: Gewählte Hardwareeigenschaften des GPGPU-Rechenbeschleunigers.

Hardwaregrößen wie etwa die Anzahl an Rechenkernen, Hardware-Threads, sowie der L1i/d-

und L2-Cache vorab für alle Modelle festgelegt (Tabelle 7.4).

Als Randnotiz soll erwähnt sein, dass bei jedem Start des GPGPU-Rechenbeschleunigers zuerst eine Instruktionsbootsequenz vor der *main()*-Prozedur und somit vor der jeweiligen Anwendung ausgeführt wird, bei welcher *Heap* und *Stack* eingerichtet werden. Die Anzahl der benötigten Instruktionen ist jedoch konstant und äußerst gering. Nach Bearbeiten des Bootcodes springt ein jeweiliger Hardware-Thread direkt in *main()*, wodurch die Anwendung *bare-metal* betrieben wird und kein Betriebssystem zu Irritationen (Stichwort: *jitter*) führen kann.

Prozessor	Intel Core i7-7500U, 2.7-3,5 GHz
Speicher	16 GB DDR4-2133 + 8 GB DDR4-2133
Betriebssystem	Arch Linux (4.11.9-1)
Compiler	g++ 7.1.1 20170630
Compileroptionen	-Ofast -march=native

Tab. 7.5: Die zur Messung des GPU-Simulators verwendete Testplattform.

Die für das Experiment gewählte Evaluierungsplattform ähnelt der Plattform für die Evaluierung des HMC-Simulationsmodells (Tabelle 7.5).

7.3.1 Evaluation der Grafikausgabe

Angelehnt an das Emulator-Modell wurde das GPU-Simulationsmodell mit einem Grafikspeicher ausgestattet. Die tatsächliche Grafikausgabe übernimmt dabei die Bibliothek *Simple DirectMedia Layer* (SDL) in der Version 2, welche den simulativen Bildspeicherinhalt entsprechend als Bild mit dynamischen Inhalt rendert. Zugleich wurde der Bildspeicher an die Adresse im Adressraum gelegt, an welcher dieser auch in der realen Hardware wiederzufinden ist, sodass *NYUZI*-Anwendungen binärkompatibel sind.

Das Simulationsmodell selbst wurde in die ganzheitliche virtuelle Plattform (VP) *SoCRocket* eingebunden, damit die gewählte Architektur weniger als Rechenbeschleuniger sondern hierbei primär als Grafikbeschleuniger evaluiert werden kann (Unterabschnitt 6.2.3). Vor Ablauf des Experiments kopiert der Hauptprozessor die Instruktionen sowie Daten in den eigenständigen Grafikspeicher. Nach Abschluss dessen wird die Berechnung per Delegation gestartet, woraufhin der Hauptprozessor eine Warteschleife ausführt, um auf das Endresultat der Grafikausführung zu warten.

Unter vielen verfügbaren Anwendungen befindet sich die Anwendung *Rotozoom*. Diese wurde schlicht als Machbarkeitsstudie genutzt, da diese einerseits die Einbettung innerhalb der *SoCRocket*-Plattform sowie andererseits die Bildschirmausgabe aufzeigen kann. Im Detail beschreibt diese Anwendung den Bildschirminhalt, entsprechend den Bildspeicher mit aneinander gereihten und gleichzeitig aufeinander gestapelten Smilies. Diese drehen sich mit jedem Zeitschritt um einen gedachten Bildmittelpunkt, wobei gleichzeitig in das Bild hineingezoomt wird (Abbildung 7.7).

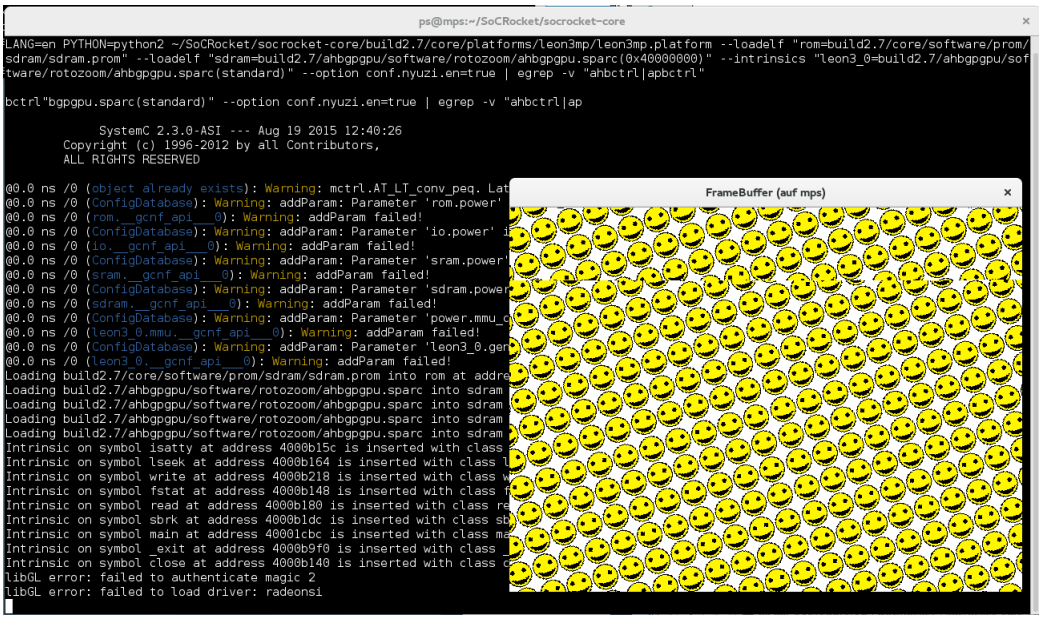


Abb. 7.7: Ausführung der Anwendung *Rotozoom* auf der virtuellen Plattform *SoCRocket* mit zugeschaltetem GPU-Simulationsmodell und *SDL*-Grafikausgabe.

7.3.2 Simulationsgenauigkeit bei variierenden Speicherlatenzen

Das GPU-Simulator ist ohne DRAM-Speichermodell entwickelt worden, bietet jedoch die Möglichkeit einer statisch konfigurierbaren Speicherlatenz. Mithilfe derer ist eine Approximation des Modells möglich, sodass die Simulationsgenauigkeit ermittelt werden kann. Obgleich insgesamt drei Modelle, namentlich Emulator, Simulator und Verilator (Version 3.902-1) verfügbar sind, um *NYUZI*-Anwendungen auszuführen, wird der Emulator in diesem Fall nicht in Erwägung gezogen, bietet dieser doch nur ein funktionales und somit kein taktgenaues Modell.

Für die weitere Evaluierung der Simulationsgenauigkeit wird auf die seismische RTM mit 9-Punkt-*Stencil* zurückgegriffen, welche mit unterschiedlichen Problemgrößen, namentlich Breite und Höhe sowie Zeitschritte, evaluiert wurde (Abschnitt 7.2). Bedingt durch die genutzte Vektoreinheit muss die Breite ein Vielfaches von 64 Byte betragen, damit die sechzehn parallelen Fließkomma-ALUs genutzt werden können. Weitere Einschränkungen sind jeweils vier zusätzlich benötigte Reihen (hinsichtlich der Breite) sowie eine Spalte mit der Vektorbreite, welches durch die Vektorisierung des *Stencils* resultiert. Somit ergibt sich für die Breite $b(x) = 16 \cdot x + 16$ und die Tiefe $t(z) = z + 4$, wodurch für die Anwendungsausführung auf eine quasi-Quadratur d.h. auf Breite $b(N) = 16 \cdot N + 16$ sowie Tiefe $t(N) = 16 \cdot N + 4$ zurückgegriffen werden muss. Der maximale Speicherbedarf einer jeweiligen Konfiguration ist dabei $b(N) \cdot t(N) \cdot 4\,\text{Bytes}_{\text{EinfacheGenauigkeit}} \cdot 3$ (Unterabschnitt 7.2.2) Die Verdreifachung resultiert aus den Speichern APF, NPF und VEL. Hinzu kommt das *Ricker Wavelet* welches je Zeitschritt eine einfache Fließkommazahl benötigt, wodurch z.B. bei 100 Zeitschritten bereits 400 Byte anfallen.

Obgleich multiple Verzögerungszyklen als statische Speicherlatenz für den Simulator in Betracht gezogen worden sind, wurden für die Schaubilder jeweils eine große Latenz (50 Zyklen), eine kleine Latenz (30 Zyklen) und eine mit geringer Abweichung (39 Zyklen) zum tatsächlichen Verilator-Modell illustriert (Abbildung 7.8, Abbildung 7.9). Die beiden aufgezeigten Evaluierungen wurden mit 100 und 200 Zeitschritten betrieben und zeigen repräsentativ, dass die Anzahl

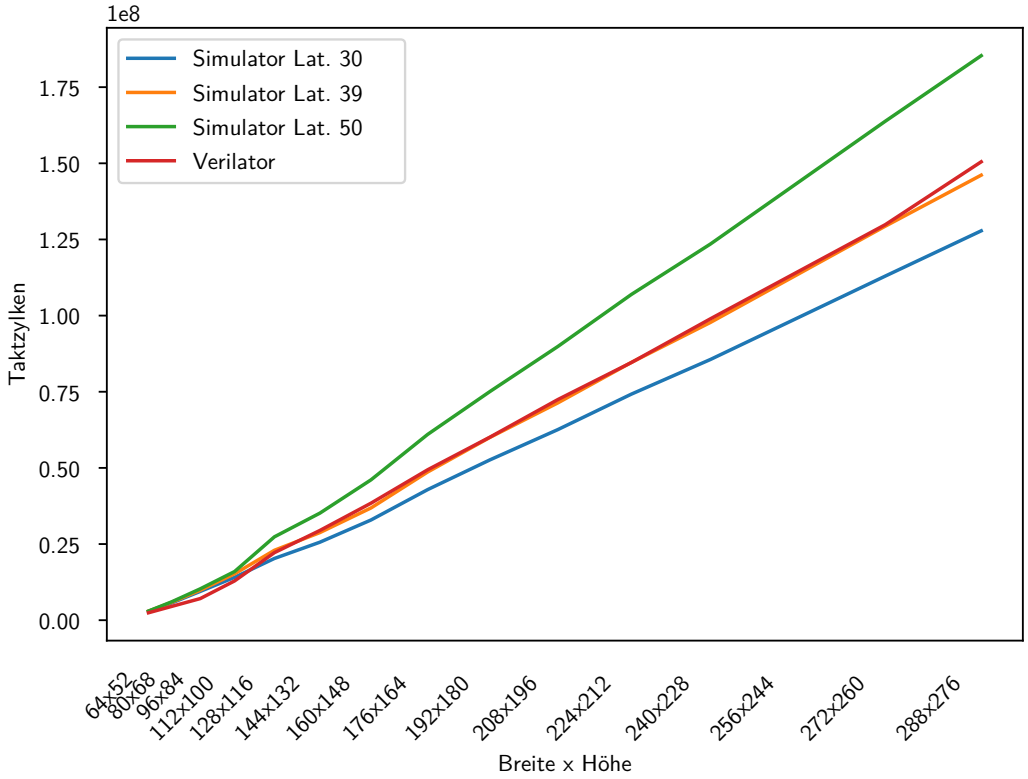


Abb. 7.8: Benötigte Taktzyklen bei Ausführung des seismischen RTM-Algorithmus. Zeitschritte: 100.

der Zeitschritte sich kaum – wie zu erwarten – auf die Abweichung zwischen Simulator und Verilator-Modell auswirkt.

Bei Bildgrößen kleiner gleich 144x132 lassen sich zwischen optimal gewählter Speicherlatenz und dem tatsächliche Hardware repräsentierenden Verilator-Modell geringfügige Unterschiede bei der Anzahl benötigter Taktzyklen ausmachen. Jedoch approximiert der Simulator die Menge an benötigten Taktzyklen ab einer Bildgröße von 144x132 recht genau. Es ist zu vermuten, dass 144x132 die Grenze darstellt, an welcher der auch im Simulator modellierte L2-Cache überschritten wird, jedoch findet dies bereits bei 112x100 (131,25 KB, bzw. 132,65 KB bei 100 Zeitschritten) statt. Somit wirkt sich die statische Latenz bei den kleinen Bildgrößen stärker aus, da hier der L2-Cache nur einmalig gefüllt wird. Werden jedoch Daten wieder aus dem L2-Cache verdrängt, was insbesondere bei größeren Bildgrößen von statten geht, so pendelt sich das Verilator-Modell um die statische Latenz von 39 Zyklen ein.

Wird die Speicherlatenz von 39 Zyklen für einen direkten Vergleich zwischen Verilator- und Simulator-Modell herangezogen, so ergibt sich eine Abweichung von 6,1 % (100 Zeitschritte) bzw. respektive 5,7 % (200 Zeitschritte). Die Abweichung wirkt in Angesicht der gegebenen Messungen recht hoch, überlappt doch die Kennlinie des Verilator-Modell mit der des mit 39 Zyklen Speicherlatenz betriebenen Simulators, wobei dies jedoch Durchschnittswerte darstellen. Wird alleine der Bereich von 64x52 bis 144x132 in Erwägung gezogen, so ergibt sich eine Abweichung von 15 % wohingegen alles nach 144x132 gegen 1 % strebt. Obgleich bewusst, fällt die Abweichung für die beiden Fälle mit einer Speicherlatenz von 30 Zyklen und 50 Zyklen mit 11 % bzw. 50 Zyklen deutlich signifikanter aus. Es resultiert, dass für eine gute Approximation des Verilator-Modells

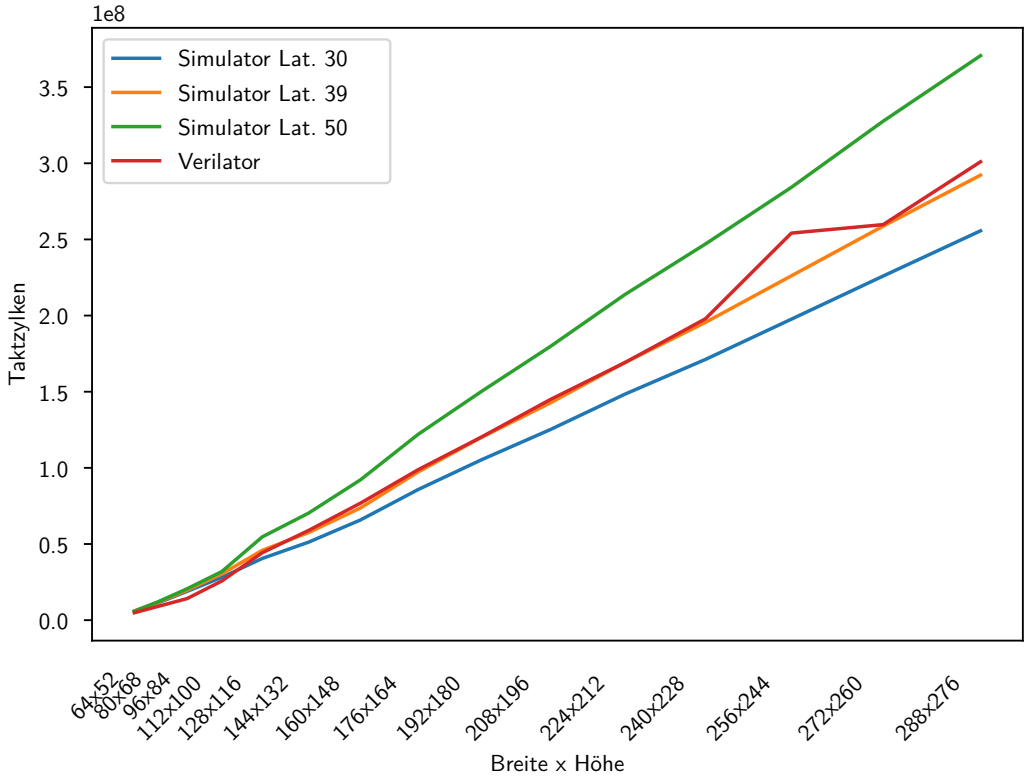


Abb. 7.9: Benötigte Taktzyklen bei Ausführung des seismischen RTM-Algorithmus. Zeitschritte: 200.

eine Speicherlatenz von 39 Zyklen nutzbar ist, um eine Abweichung von ca. 6 % zu erzielen.

7.3.3 Detailbetrachtung der Simulationsgenauigkeit

Wird ein Simulator gegen die tatsächliche Hardware – hiermit repräsentiert durch das Verilator-Modell – verglichen, sind insbesondere die Anzahl an ausgeführten Instruktionen von Interesse. Sollten diese bereits signifikant abweichen, stellt dies die Aussagekraft des Simulators in Frage. Auf Basis der zuvor evaluierten statischen Speicherlatenz von 39 Zyklen soll die Problemgröße von 176x164 mit jeweils 100 sowie 200 Zeitschritten vertieft betrachtet werden (Unterabschnitt 7.3.2). Zwei Tabellen weisen die aggregierten Messergebnisse eines modellierten GPU-Rechenkerns aus, welche wiederum mit jeweils vier Hardware-Threads betrieben wurde (Tabelle 7.6, Tabelle 7.7). Werden insbesondere die ausgeführten Instruktionen verglichen, so weichen diese nur minimal bzw. im Detail konstant um knapp 64 Instruktionen ab, was auf die Art wie die Simulation gestartet und beendet wird, zurückzuführen ist. Ähnlich schneiden die benötigten Zyklen ab, welche nur unwesentlich abweichen. Es zeigt sich, dass bei einer Verdopplung der Zeitschritte ebenfalls eine Verdopplung der nötigen Zyklen, Instruktionen als auch Messwerte hinsichtlich der Cachehierarchie vonstatten geht. Dies ist einerseits bei beiden Modellen gleich und bewegt sich zugleich im gleichen Rahmen.

Trotz der kleinen Kapazität von 16 KB bietet der L1i-Cache laut Messung genügend Platz, um den seismischen RTM-Algorithmus mit hoher Trefferquote zu bedienen. Gleiches spiegelt sich mit der Zunahme der Zeitschritte wieder. Einzig der L2-Cache weist eine starke Diskrepanz

Zeitschritte	100		
Typ	Entwickelter Simulator	Verilator	Abweichung [%]
$\sum$ [Zyklen]	48.655.541	49.378.378	1,5
Erstellte Instruktionen	8.981.609	8.185.569	8,9
Ausgeführte Instruktionen	7.219.251	7.219.297	0,0
L1i-Misses	64	63	1,6
L1i-Hits	14.085.223	14.367.271	2,0
L1d-Misses	804.238	804.220	0,0
L1d-Hits	1.124.330	1.124.330	0,0
L2-Misses	869.003	567.574	65,3
L2-Hits	95.526	396.936	415,5

Tab. 7.6: Erzielte Ergebnisse von Simulator als auch Verilator bei einer seismischen Problemgröße von 176x164 und 100 Zeitschritten. Beide waren nach Tabelle 7.4 konfiguriert, wobei der Simulator eine konstante Speicherlatenz von 39 Zyklen nutzt.

Zeitschritte	200		
Typ	Entwickelter Simulator	Verilator	Abweichung [%]
$\sum$ [Zyklen]	97.291.541	98.758.275	1,5
Erstellte Instruktionen	17.962.509	16.370.268	8,9
Ausgeführte Instruktionen	14.437.851	14.437.896	0,0
L1i-Misses	64	63	1,6
L1i-Hits	28.168.923	28.732.162	2,0
L1d-Misses	1.608.438	1.608.420	0,0
L1d-Hits	2.248.530	2.248.530	0,0
L2-Misses	1.737.453	1.135.283	65,3
L2-Hits	191.376	793.527	414,6

Tab. 7.7: Erzielte Ergebnisse von Simulator als auch Verilator bei einer seismischen Problemgröße von 176x164 und 200 Zeitschritten. Beide waren nach Tabelle 7.4 konfiguriert, wobei der Simulator eine konstante Speicherlatenz von 39 Zyklen nutzt.

zwischen Simulator und Verilator aus. Beide Werkzeuge verzeichnen eine höhere Anzahl von Misses als Hits innerhalb ihres L2-Caches, jedoch widerfuhren dem Simulator für beide Beispiele mehr Misses als Hits im Gegensatz zum Hardware-Modell. Ein Grund hierfür stellt die Anbindung des L2-Caches an den Hauptspeicher dar, bei welcher das Simulationsmodell rein auf die konstante Speicherlatenz zurückgreift, wohingegen der Verilator einen tatsächlichen AXI-Bus mit tatsächlichem DRAM-Speicher modelliert. Zusätzlich könnte die Ersetzungsstrategie innerhalb des L2-Caches unterschiedlich sein, sodass hierauf für zukünftige Arbeiten ein Augenmerk geworfen werden sollte.

Trotz Diskrepanz hinsichtlich der Ergebnisse im L2-Cache stellt dies jedoch keinen Nachteil dar, da der genutzte Algorithmus zu der Klasse der Strömungsalgorithmen zählt und dieser bei groß gewählten Problemstellungen ohnehin nicht oder nur eingeschränkt von Caches profitieren kann. Somit zeigt bereits dieses Beispiel anhand einer Problemgröße von 176x164 und 100 sowie 200 Zeitschritten, dass der Simulator die Fähigkeit besitzt, recht zuverlässig und akkurat die Hardwarecharakteristik – insbesondere die Rechenpipeline – der *NYUZI*-Architektur zu approximieren.

7.3.4 Evaluation der Simulationsgeschwindigkeit

Die Sinnhaftigkeit der manuellen Entwicklung eines Simulators insbesondere im Hinblick auf die Existenz eines Verilator-Modells, welches hardwaregetreu die SystemVerilog-Beschreibung in C++ übersetzt und daraufhin kompiliert, kann anhand der Ausführungsgeschwindigkeit bemessen werden. Obgleich letzteres ebenfalls als Modell auf einer Zielhardware – angeblich schnell – simuliert werden kann, kann ein Simulator abstrakter dargestellt und somit auf Simulationsgeschwindigkeit hin optimiert werden. Für die Evaluierung wurde wie zuvor die Evaluierungsplattform Tabelle 7.5 herangezogen, wobei für den Simulator auf die bekannten Speicherlatenzen von 30, 39 und 50 Zyklen zurückgegriffen wurde.

Wird rein der Emulator betrachtet, bildet dieser eine nahezu horizontale lineare Ausführungsgeschwindigkeit (Abbildung 7.10, Abbildung 7.11). Da der Emulator rein funktional ausgeführt wird, somit keine Taktinformationen zur Verfügung stellt, soll dieser als maximale Referenzsimulationsgeschwindigkeit angesehen werden. Alle gewählten seismischen Problemstellungen – in Form von Bildgrößen – werden durch das Emulator-Modell innerhalb weniger Sekunden bearbeitet. Im Gegensatz dazu benötigt das Verilator-Modell bereits bei kleinen Bildgrößen signifikant mehr Zeit, sodass selbst die Berechnung der winzigen Bildgröße von 240x228 bei 100 Zeitschritten nahezu zehn Minuten benötigt. Werden die Zeitschritte verdoppelt, verdoppelt sich entsprechend auch die Simulationszeit.

In Anbetracht der Messung setzt sich das entwickelte Simulationsmodell klar vom Verilator-Modell ab, obgleich ersteres dabei nahezu taktgenau simuliert. Dies ist insbesondere beachtlich, da die benötigte Simulationszeit sich im Bereich des Emulator-Modells bewegt, wohingegen dieses wiederum rein funktional die GPU simuliert. Hinzu kommt, dass die gewählte Speicherlatenz dabei unabhängig ist. Wird die Problemgröße anhand der Zeitschritte von 100 auf 200 vergrößert, verdoppelt sich zwar für alle Modelle die Ausführungszeit entsprechend, jedoch fällt dies seitens des Simulationsmodells nicht ins Gewicht. Für weitere Experimente stellt somit das entwickelte GPU-Simulationsmodell eine ideale Ausgangssituation dar, ist doch die allgemeine Abweichung von 6 % in Kombination mit einer äußerst schnellen Simulationsgeschwindigkeit gerade für Ent-

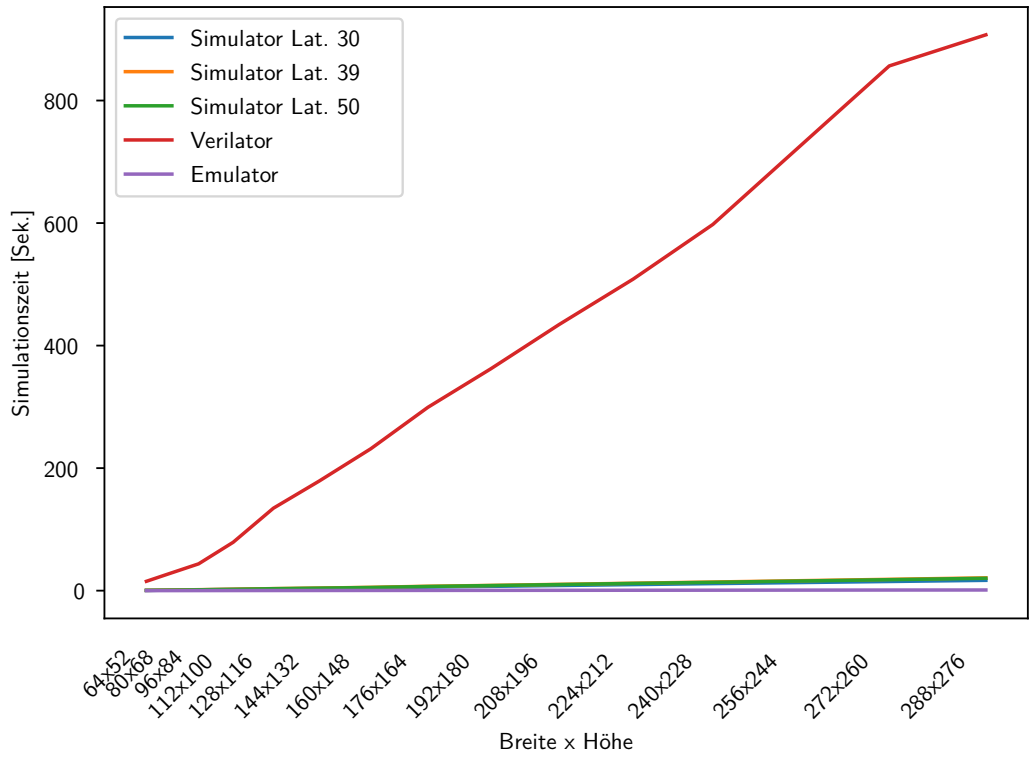


Abb. 7.10: Benötigte Simulationszeit bei Ausführung des seismischen RTM-Algorithmus. Zeitschritte: 100.

wurfsraumexplorationen von höchstem Interesse.

7.4 Dreidimensional integrierter PIM-Beschleuniger mit GPU-Rechenkernen

Unterabschnitt 6.3.2 schildert eine PIM-Ausgangsbeschleunigerarchitektur, bei welcher ein standardkonformer HMC-Speicher mit einer optimierten GPU-Architektur vernetzt wurde (Abschnitt 6.3). Obgleich die gewählten GPU-Architekturverbesserungen in der Theorie höhere Fließkommarechenleistung erzielen sollten, so werden diese als Gesamtkonstrukt im Folgenden evaluiert. Hierzu werden verschiedene Szenarien betrachtet. Einerseits wird ein einzelner optimierter GPU-Rechenkern mit einer unterschiedlichen Anzahl an Hardware-Threads an einem idealisierten HMC betrieben. Andererseits wird das gleiche Experiment mit mehreren Rechenkernen betrachtet.

Idealisiert ist in diesem Sinne ein HMC, welcher volle Speicherbandbreite liefert und keine Latenz der GPU-Architektur ausweist. Dadurch lässt sich die maximal abrufbare Speicherbandbreite des GPGPU-Rechenbeschleunigers in Kombination mit dem gewählten seismischen Reverse Time Migration ermitteln (Abschnitt 7.2). Wird dabei nur ein Rechenkern betrachtet, wird voraussichtlich die notwendige Synchronisierung durch die Hardwarebarriere weniger gravierend ausfallen, als wenn mehrere Rechenkerne diese nutzen müssen. Zugleich zeigt sich, ob mehrere Hardware-Threads zugleich die Rechenpipeline effizient nutzen können.

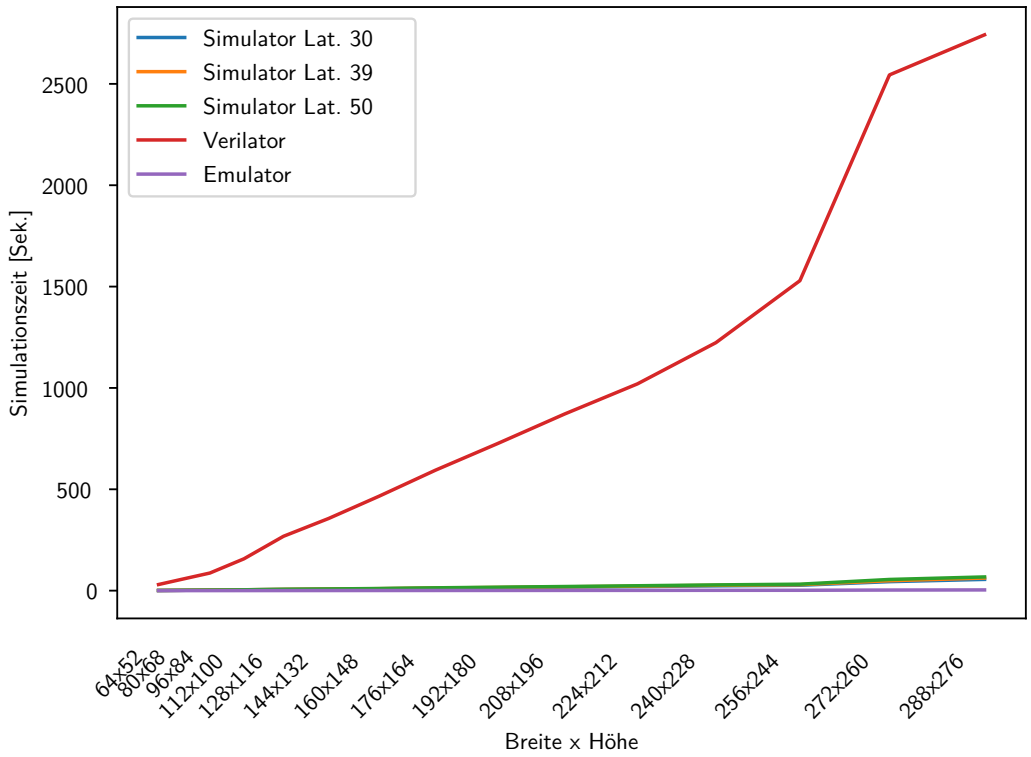


Abb. 7.11: Benötigte Simulationszeit bei Ausführung des seismischen RTM-Algorithmus. Zeitschritte: 200

Mit den gemachten Erkenntnissen soll daraufhin ein tatsächliches HMC-Modell mit entsprechenden Latenzen in Betracht gezogen werden. Bevor jedoch die PIM-Beschleunigerarchitektur evaluiert wird, soll vorab die Adressverwürfelung über die *Vaults* betrachtet werden.

7.4.1 Effekt der Adressverwürfelung

Um die größtmögliche Speicherbandbreite aus einem HMC-Speicherstapel abzurufen, muss die Adresszuordnung mit bedacht werden. Hierzu ist es wichtig, dass die Abbildung der *Vaults* nahezu parallel bedient wird, da *Vaults* unabhängig voneinander jeweils 8 Byte d.h. in Summe 256 Byte pro Takt zur Verfügung stellen. Vorausgesetzt ein standardkonformer HMC mit

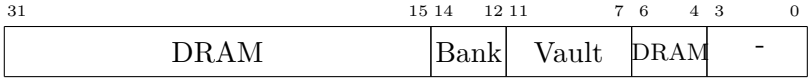


Abb. 7.12: Standardadresszuordnung eines mit der Spezifikation 2.1 konformen und 4 GB großen HMC-Speicherstapels bei einer Blockgröße von 128 Byte.

einer Kapazität von 4 GB und einer Blockgröße von 128 Byte wird für die weiteren Experimente verwendet, so erfolgt die *Vault*-Adressierung per siebtem bis elftem Bit (Abbildung 7.12). Entsprechend wird bereits alle 4 KB rotiert d.h. wieder von null angefangen. Ebenso wird alle 128 Byte ein weiterer *Vault* adressiert.

Da bereits mit einer Problemgröße von 4 KB alle *Vaults* zumindest einmal beansprucht

worden sind, können bereits kleine Problemgrößen in Kombination mit in Versatz startende Hardware-Threads die maximale Speicherbandbreite abrufen. Positiv beeinflusst dies die Zugriffsgranularität, entsprechend Zeilengröße, des L2-Caches, welcher auf 128 Byte gesetzt wurde. Entsprechend wird mit hoher Wahrscheinlichkeit jede weitere eingeladene Cachezeile durch die sequentielle Ausführung des Algorithmus auf den folgenden *Vault* zugeordnet. Abbildung 7.13

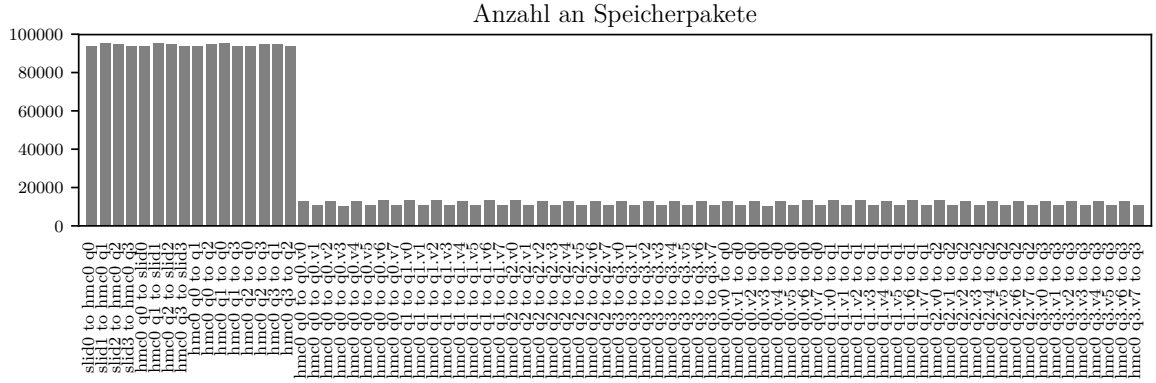


Abb. 7.13: Verbindungsbeanspruchung des RTM-Algorithmus bei einer Problemgröße von 516x516 bei fünf Zeitschritten.

soll hierzu als Beispiel dienen, welches die annähernde Gleichverteilung der Speicherzugriffe über die *Vaults* bei einer Problemgröße von knapp 1 MB aufzeigt. Grund der leicht versetzten Peaks kann sich aufgrund von unvollständiger Ausnutzung von Seitenbereichen des seismischen RTM-Algorithmus sowie die zu ladenden Instruktionen auftun, wobei ersteres durch die *Linker*-Speicherplatzierung und dessen Auswirkung auf die *Vaults* entstehen kann.

7.4.2 PIM-Ausgangsbeschleunigerarchitektur mit idealisiertem HMC

Für alle nachfolgenden Experimente wurde das PIM-Simulationswerkzeug auf einen HMC mit einer Kapazität von 4 GB eingestellt (Tabelle 7.8). Zugleich wurde die Standard-Blockgröße innerhalb des HMCs auf 128 Byte gesetzt. Da in dem ersten Experiment ausschließlich ein Rechenkern untersucht werden soll, wurde auf eine einzelne *Lane* zurückgegriffen, welche direkt mit dem GPU-Rechenkern verbunden ist. Entsprechend stehen diesem 60 GB/s bidirektional d.h. in beide Richtungen gleichzeitig zur Verfügung. Um zu untersuchen, inwieweit ein einzelner, optimierter GPU-Rechenkern die Speicherbandbreite bei der Ausführung des seismischen RTM-Algorithmus konsumieren kann, wurde ein idealisierter HMC in Erwägung gezogen, welcher mit einer Latenz von null Taktzyklen die volle Bandbreite zur Verfügung stellen kann. Entsprechend werden interne HMC-(Schnittstellen)-Konflikte, welche aus einem Ring oder aus einer Kreuzschienenverteilung (engl.: *crossbar*) resultieren können, ausgeblendet. Auch wird keine Betrachtung von mehreren GPU-Rechenkernen erwogen, da die gemeinsame Ressource, der HMC, idealisiert dargestellt sind. Somit würde bei einer Konfiguration mit mehreren GPU-Rechenkernen rein die Barriere als unmittelbarer Beeinflusser in Frage kommen, sollte jedoch bedingt durch deren spezieller Entwurf keine große Auswirkung auf das Gesamtergebnis haben.

Hinsichtlich der Cachehierarchie wird auf die optimierte, d.h. auf den L1i-Cache mit einer Kapazität von 1 KB und auf den L2-Cache mit einer Kapazität von 8 KB zurückgegriffen. Der L1d entfiel, entsprechend existiert je Hardware-Thread eine tiefe Lade-/Speicherwarteschlange.

Rechenkerne	1
Threads	4
L1i-Kapazität	1 KB
L1i-Blockgröße	64 Byte
L1i-Blöcke	4
L1i-Sätze	2
L2-Kapazität	8 KB
L2-Blockgröße	256 Byte
L2-Blöcke	16
L2-Sätze	2
L2-Strategie	*Write-Back*
HMC-SLIDs	1
HMC-SLID-Bandbreite	aggregiert 120 GB/s (60 GB/s je Richtung)
HMC-Latenz	0 Takte
HMC-Kapazität	4 GB
HMC-Partitionsblockgröße	128 Byte
HMC-Quadranten-Verbindung	Ringbus
HMC-Quadranten-Bandbreite	aggregiert 120 GB/s (60 GB/s je Richtung)

Tab. 7.8: PIM-Konfiguration ausgestattet mit GPU-Rechenkernen und idealisiertem HMC (Unterabschnitt 6.3.1, Unterabschnitt 6.3.2).

In Anbetracht eines *NYUZI*-Hardwaremodell, bei welchem der GPGPU-Rechenbeschleuniger eine maximale Taktfrequenz von 570 MHz erzielen konnte, wird zur Vereinfachung die Hälfte des mit 1,25 GHz getakteten HMC, also 625 MHz angenommen (Unterabschnitt 6.2.1). Eine Beschleunigung auf diese Frequenz sollte mit moderaten Veränderungen der SystemVerilog RTL-Implementierung sowie mit Hilfe von aggressiveren Industrie-Verilog-Bibliotheken realisierbar sein.

Für die Problemstellung wird die erarbeitete seismische Reverse Time Migration herangezogen, welche mit einem Größenverhältnis von 2052x2052 und mit insgesamt fünf Zeitschritten betrieben wird (Abschnitt 7.2). Dies stellt eine starke Skalierung (engl.: *strong scaling*) dar, da bei der Zunahme an Rechenkernen die Problemstellung jeweils verkleinert über diese verteilt wird (Unterabschnitt 2.1.15). Nicht weiter aufgeführt ist eine schwache Skalierung des seismischen RTM-Algorithmus, bei welcher die Problemgröße bei der Zunahme von Rechenkernen jeweils vergrößert wird. Eingehende Untersuchungen zeigten auf, dass dies nur minimale Beschleunigungseffekte auf die Gesamtrechengeschwindigkeit ergab.

Bedingt durch die geringen Cache-Größen und da dieses Breite-/Höhe-Verhältnis bereits bei einfacher Fließkommagenauigkeit knapp 16 MB an Daten produziert, sollten sich entsprechende Verdrängungseffekte innerhalb der Cachehierarchie einstellen. Insgesamt sollten in etwa 29,3 MFLOP $(= (2052_{Höhe} - (2 \cdot 2)_{Ränder}) \cdot (2052_{Breite} - (2 \cdot 2)_{Ränder}) \cdot 14\,\text{FLOP}_{Stencil} \cdot 5_{Zeitschritte})$ für die gewählte Problemstellung berechnet werden (Unterabschnitt 7.2.3). Hinsichtlich des Compilers wurde rein die Option `-O3` aktiviert, wobei der innerste *Stencil*-Kernel zweifach ausgerollt wurde.

Ausgangsevaluation der PIM-Beschleunigerarchitektur

Im Anbetracht der Gleichverteilung der Problemstellung über alle Hardware-Threads hinweg, kann davon ausgegangen werden, dass die auszuführenden Instruktionen bei allen Hardware-Threads in etwa gleich ausfallen werden. Auch die benötigten Taktzyklen sollten gleich ausfallen, da die eingebrachte Höchstleistungsbarriere für eine äußerst präzise Synchronisation und somit zu wenig Mehraufwand führt. Beides wird durch das Ergebnis bestätigt (Tabelle 7.9). Wie vor-

Rechenkern	0			
Thread	0	1	2	3
Taktzyklen	61.114.665			
Instr. ausgeführt	12.319.081	12.319.073	12.319.073	12.319.073
Instr. dekodiert	12.351.623	12.384.301	12.374.439	12.377.496
Vektor-FP-Instr.	4.596.480	4.596.480	4.596.480	4.596.480
Pipeline-Auslastung [%]	80,98			
Rechenleistung [GFLOPS]	3,01			
L1i Hits	53.465.759			
L1i Misses	56			
L2 Hits	2.295.390			
L2 Misses	8.038.975			
L2 Stalls	0			
L2 Writebacks	1.019.965			
Speicherbandbreite [GB/s]	25,37			

Tab. 7.9: Simulationsergebnis der PIM-Beschleunigerarchitektur, bei welcher ein GPU-Rechenkern mit vier Hardware-Threads bei 625 MHz (Tabelle 7.8) einen seismischen RTM-Algorithmus mit 9-Punkt-*Stencil* bei einer Problemstellung von 2052x2052 und fünf Zeitschritten berechnet.

ab festgestellt, berechnete ein Rechenkern knapp 29,4 MFLOP ($= 4.596.480_{Vektor-FP-Instr.} \cdot 4_{Threads} \cdot 16_{Vektorbreite}$), sodass das evaluierte Resultat zugleich stimmig ist. Per se werden etwas mehr FLOP berechnet, da je Zeitschritt das *Ricker Wavelet* mit einer Addition je Zeitschritt zu Tragen kommt und da der Compiler Fließkommaoperationen wahlfrei einfügen kann. Wie sich zeigt, findet wie erwartet gerade bedingt durch die gewählte Problemstellung eine hohe Verdrängung in dem L2-Cache statt.

Etwas überraschend ist die bereits hohe Auslastung von knapp 81 %. Damit einher geht eine Fließkommaleistung von 3 GFLOPS, welche im Rahmen von maximal zehn erzielbaren GFLOPS einer Effizienz von 30 % gleich kommt (Tabelle 6.2). Im Kontrast zu Tsuboi et al., welche eine Effizienz von insgesamt 12 % für seismische Algorithmen ausweisen, ist dieses Ergebnis durchaus beachtlich [Tsuboi et al., 2016]. Die benötigte Speicherbandbreite fällt mit 25,37 GB/s noch gering aus, bietet doch eine HMC-*Lane* 60 GB/s pro Richtung.

Effekt der Taktfrequenz auf die Speicherbandbreite

Werden mehr Hardware-Threads herangezogen, konsumieren diese zwar zugleich mehr Speicherbandbreite (Abbildung 7.14), jedoch reizt der GPGPU-Rechenbeschleuniger mit einer Taktfre-

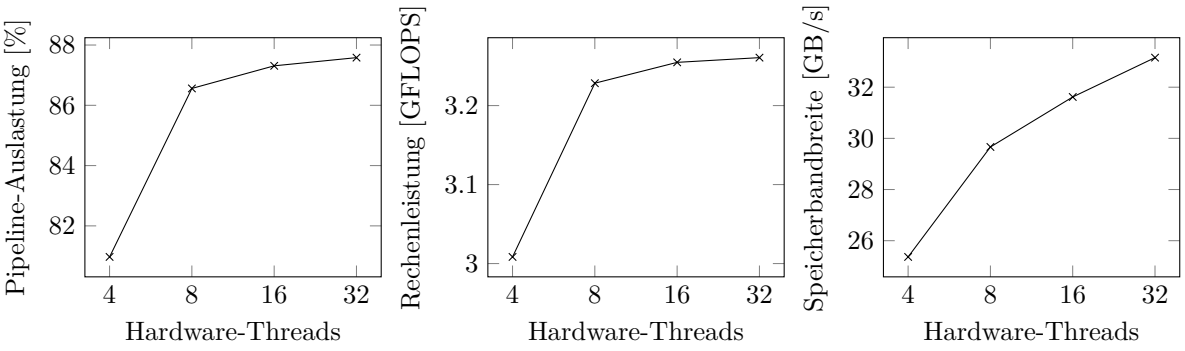


Abb. 7.14: Effekte aus der Skalierung der Hardware-Threads eines einzigen GPU-Rechenkerns, welcher mit 625 MHz innerhalb eines PIMs mit idealisiertem HMC betrieben wird (Tabelle 7.8) und einen seismischen RTM-Algorithmus mit 9-Punkt-*Stencil* bei einer Problemstellung von 2052x2052 mit fünf Zeitschritten berechnet.

quenz von 625 MHz noch immer nicht die maximale Speicherbandbreite der HMC-*Lane* aus. Es zeigt sich, dass bedingt durch die bereits hohe Auslastung der Verarbeitungspipeline diese mit Zunahme der Hardware-Threads kaum mehr ausgelastet werden kann (bis zu 6 %). Gleicher Effekt ist bei der Rechenleistung wahrnehmbar, welche sich um 10 % auf bis zu 3,3 GFLOPS für den einzelnen Rechenkern verbessert. In Summe ist eine Verdopplung der Hardware-Threads von vier auf acht durchaus plausibel, wohingegen eine weitere Verdopplung von acht auf sechzehn sowie sechzehn auf 32 kaum mehr von Profit ist.

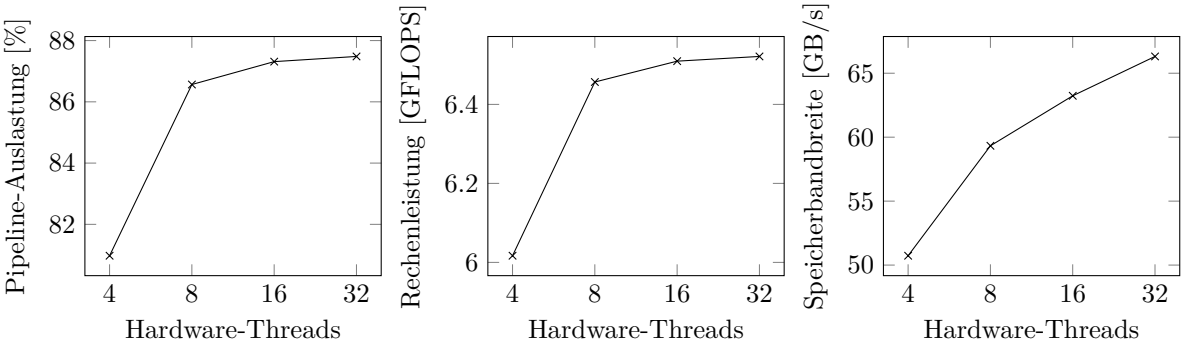


Abb. 7.15: Effekte aus der Skalierung der Hardware-Threads eines einzigen GPU-Rechenkerns, welcher mit 1,25 GHz innerhalb eines PIMs mit idealisiertem HMC betrieben wird (Tabelle 7.8) und einen seismischen RTM-Algorithmus mit 9-Punkt-*Stencil* bei einer Problemstellung von 2052x2052 mit fünf Zeitschritten berechnet.

Ein ähnliches zu dem soeben beschriebenen Bild zeigt sich, wenn die Taktfrequenz von 625 MHz auf 1,25 GHz angehoben wird (Abbildung 7.15). Im Detail verändert sich nichts an der Auslastung der Rechenpipeline, jedoch verdoppeln sich nun die Fließkommarechenleistung sowie der Bedarf an kumulierter Speicherbandbreite. Somit ist die PIM-Beschleunigerarchitektur mit 625 MHz klar limitiert durch die Taktfrequenz. Auch der AMC wird gleichsam mit 1,25 GHz betrieben, sodass die gleiche Taktfrequenz für HMC und GPU innerhalb eines PIMs durchaus als realistisch anzusehen ist. Es erscheint, als sei die PIM-Beschleunigerarchitektur weiterhin durch den seismischen RTM-Algorithmus limitiert, wodurch auch eine Verdopplung der Taktfrequenz und trotz Verdopplung der erzielten Fließkommarechenleistung nur knapp 30 % der

nun 20 GFLOPS an verfügbarer Rechenleistung abgerufen werden können. Hinzu kommt nun, dass bei der Konfiguration mit 32 Hardware-Threads eine Speicherbandbreite von 66,31 GB/s benötigt wird, welche sich im Detail auf 56,75 GB/s Lesen und 9,56 GB/s Schreiben verteilt. Im Gegensatz zur Taktfrequenz von 625 MHz stellt dies nun einen Bereich dar, bei welchem durchaus eine einzige *Lane* d.h. externe Schnittstelle eines tatsächlichen HMC (im Kontrast zu dem hierbei idealisiert genutzten HMC) mit in Summe aggregierten 120 GB/s vollduplex bzw. 60 GB/s in eine Richtung gerade bei der Lesebandbreite nahezu an ihre Grenzen stößt (Tabelle 6.4).

Einfluss der Assoziativität des L2-Caches

Da nur geringfügig SRAM zur Verfügung steht, wurde bis dato auf eine Cachehierarchie mit den Größen L1i 1 KB (Blöcke: 4, Sätze: 2) und L2 8 KB (Blöcke: 16, Sätze: 2) zurückgegriffen. Der L1d-Cache hingegen wurde zur Leistungsoptimierung eingespart und gegen eine tiefe Lade-/Speicherwarteschlange ausgetauscht. An sich benötigt der L1i-Cache keine Anpassung der Assoziativität, zeigt doch nicht nur das Simulationsergebnis aus Tabelle 7.9 auf, dass bis auf wenige Anfragen alle anderen einen Treffer innerhalb des Speichers darstellen. Somit bedarf es ausschließlich der Betrachtung des L2-Caches hinsichtlich dessen Assoziativität, d.h. der Sätze und Blöcke. Vorab soll angemerkt werden, dass der gewählte Strömungsalgorithmus typischerweise keine Daten wiederverwendet, sodass ein hoher Druck auf der Cachehierarchie herrscht. Dies spiegelt sich in den Ergebnissen wieder, bei welchen unabhängig von dem gewählten Ver-

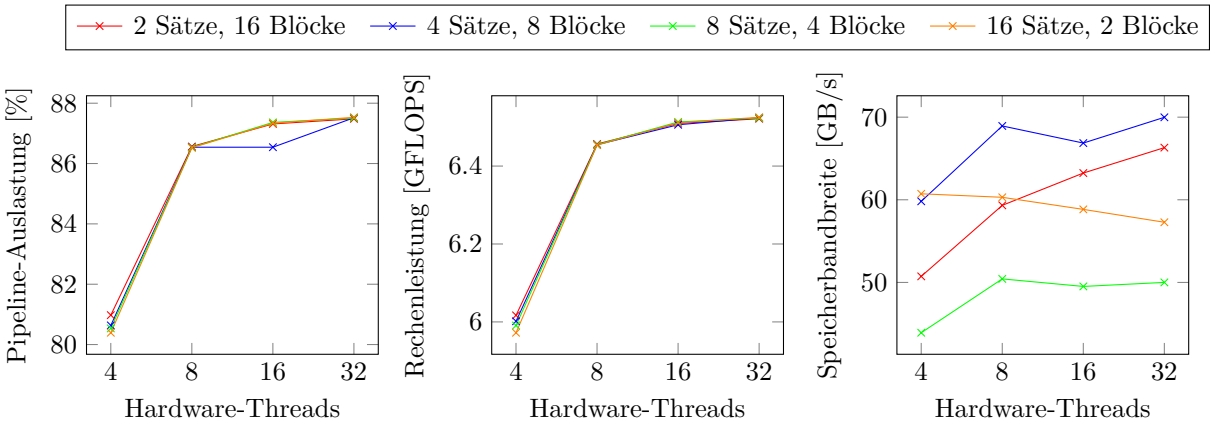


Abb. 7.16: Effekte aus der Skalierung der Hardware-Threads sowie der Modifikation der Assoziativität des L2-Caches. Der einzelne GPU-Rechenkern wird mit 1,25 GHz innerhalb eines PIM mit idealisierten HMC betrieben (Tabelle 7.8) und berechnet einen seismischen RTM-Algorithmus mit 9-Punkt-*Stencil* bei einer Problemstellung von 2052x2052 mit fünf Zeitschritten.

hältnis von Blöcken und Sätzen keine Rechenleistungsverbesserungen ersichtlich sind (Abbildung 7.16). Einzig die benötigte Bandbreite weist leichte Verbesserungen auf, indem diese mit acht Sätzen und vier Blöcken leicht reduziert werden kann. Da jedoch der idealisierte sowie reale HMC maximal 60 GB/s an Speicherbandbreite in eine jeweilige Richtung je externe Schnittstelle liefern kann, stellt auch der schlimmste Fall – im Detail wären dies vier Sätze, acht Blöcke sowie 32 Hardware-Threads – mit seinem 58,71 GB/s Lese- und 11,27 GB/s Schreibbandbreitenbedürfnis keine Limitierung dar. Theoretisch könnte mit dem noch geringfügig vorhandenen SRAM-Kontingent der L2-Cache auf 16 KB angehoben werden, jedoch sollte und dürfte dies aufgrund der explizit groß gewählten seismischen Bildgröße ebenfalls keinen Einfluss auf die

Rechenleistung haben (Unterabschnitt 6.3.2).

7.4.3 Evaluation des PIMs mit tatsächlichen HMC-Ressourcen

Im Kontrast zum vorherigen Unterkapitel soll nun ein eine PIM-Konfiguration untersucht werden, bei welcher die GPU-Rechenkerne den tatsächlichen internen (Schnittstellen-) Konflikten, Warteschlangen und den daraus resultierenden Latenzen ausgesetzt sind. Um das volle Potenzial eines

Rechenkerne	4
L2-Blöcke	4
L2-Sätze	8
HMC-SLIDs	4
HMC-SLID-Bandbreite	aggregiert 480 GB/s (60 GB/s je Richtung je *Lane*)

Tab. 7.10: Anpassungen an der mit GPU-Rechenkernen und idealisiertem HMC ausgestatteten PIM-Konfiguration (Tabelle 7.8).

HMCs auszuschöpfen wird eine Konfiguration von vier GPU-Rechenkernen betrachtet, wobei jeder GPU-Rechenkern an eine *Lane* d.h. Schnittstelle verbunden ist (Tabelle 7.10). Da bereits Abschnitt 7.4.2 aufzeigt, dass Assoziativität des L2-Caches von acht Sätzen und vier Blöcken am wenigsten Speicherbandbreite benötigt und somit das größte Potential zur Beschleunigung des Algorithmus bietet, wird diese Einstellung für den L2-Cache und damit für die weiteren Experimente verwendet (im Kontrast zu Tabelle 7.8).

PIM-Konfiguration mit vier GPU-Rechenkernen

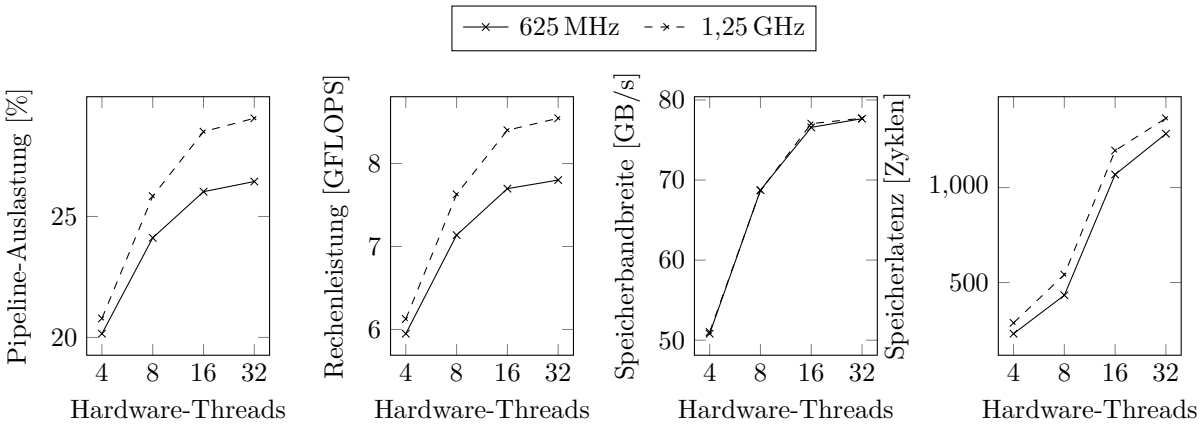


Abb. 7.17: Vier Rechenkerne sowie eine HMC-Partitionsblockgröße von 128 Byte und L2-Cacheblockgröße von 256 Byte.

Aufbauend auf Unterabschnitt 7.4.2 wurde der seismische RTM-Algorithmus mit 9-Punkt-*Stencil* mit neuer Einstellung experimentell wiederholt (Abbildung 7.17). Es zeigt sich, dass bei beiden Taktfrequenzen die Auslastung der Rechenpipeline um ein Viertel bis ein Drittel einbricht (im Kontrast zu Abbildung 7.14, Abbildung 7.15). Einen Grund hierfür stellen die vier

Rechenkerne dar, welche parallel den internen Ringbus, der die vier unabhängigen Quadranten innerhalb des HMCs verbindet, beanspruchen (Tabelle 7.8). Da die Zugriffe automatisch mit Hilfe der in der HMC-Spezifikation definierten Adressverwürfelung über die verschiedenen Quadranten und *Vaults* hinweg verteilt werden (Unterabschnitt 7.4.1), kommen unweigerlich die Inter-Quadrant-Schnittstellen zum Tragen, welches sich gerade in der maximal erzielten Speicherbandbreite widerspiegelt. Wird als Beispiel die PIM-Konfiguration mit 32 Hardware-Threads und einer Taktfrequenz von 1,25 GHz in Erwägung gezogen, so verteilen sich die erzielten 77,71 GB/s auf 58,3 GB/s Lesen und 19,4 GB/s Schreiben. In Anbetracht des internen Quadranten-Busses welcher je einen Quadranten mit einem anderen mit einer Bandbreite von 60 GB/s einfach verbindet, schöpft dies klar den internen Ringbus aus. Zugleich nehmen die Zugriffszeiten d.h. Speicherlatenzen signifikant zu, sodass selbst eine Konfiguration mit 32 Hardware-Threads – was bei vier GPU-Rechenkernen immerhin 128 parallelen Hardware-Threads in Summe entspricht – bei Zugriffszeiten von bis zu 1300 Zyklen kaum mehr Latenz-Vermeidung betreiben kann. Entsprechend sinkt auch die zu erwartende Rechenleistung signifikant, wodurch vier GPU-Rechenkerne bei verminderter Taktfrequenz von 625 MHz eine Recheneffizienz von bis zu 20 % bzw. bei 1,25 GHz ca. 11 % erzielen (Tabelle 6.2).

Verkleinerung der L2-Cacheblockgröße

Zur Optimierung des seismischen RTM-Algorithmus verwendet dieser zuvor geladene Vektorregister spaltenorientiert und nicht reihenorientiert wieder. Zugleich sind die Zeilen des L2-Caches 256 Byte breit (Blockgröße), wobei durch die Spaltenorientierung nur 64 Byte tatsächlich genutzt werden (Aufl. B.1). Die Blockgröße des L2-Caches wurde im PIM-Modellierungswerkzeug daher auf 64 Byte gesetzt, obgleich dessen Kapazität daraufhin um ein Viertel geringer ausfiel. Die HMC-Partitionsblockgröße verblieb bei 128 Byte, da diese Modifikation rein die Partitionen innerhalb des DRAMs betreffen. Anzumerken ist, dass die gewählten 64 Byte bei einem maximal möglichen HMC-Speicherzugriff von 256 Byte einen beachtlichen Mehraufwand mit sich ziehen. Grund hierfür sind die Meta-Informationen (*header*, *tail*), welche je Anfrage gleich groß sind. Tatsächlich wird zwar auf 64 Byte reduzierten Speicherzeilen jedes Datum durch den GPGPU-Rechenbeschleuniger konsumiert, jedoch resultiert daraus eine knapp 14 % geringere Netto-Speicherbandbreite. Obgleich ein signifikanter Mehraufwand vorhanden ist, kann die Aus-

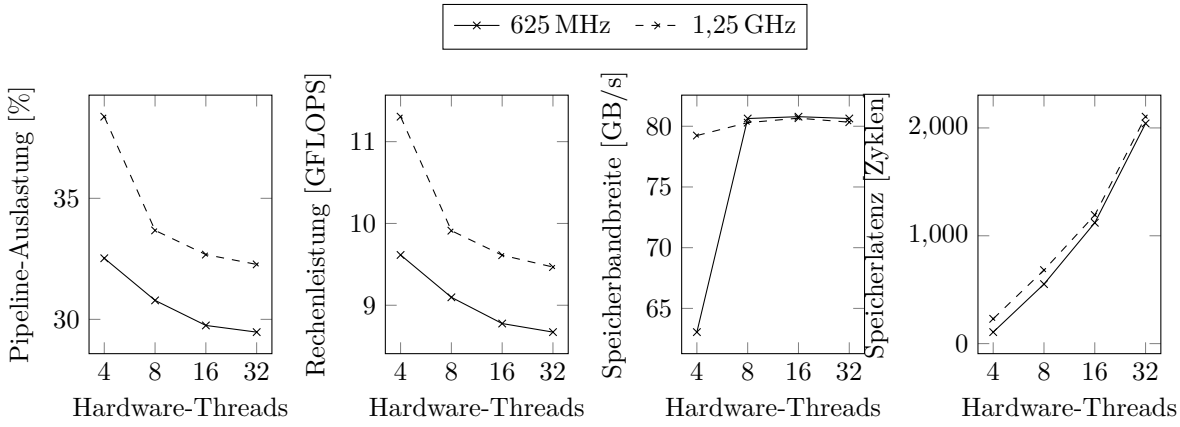
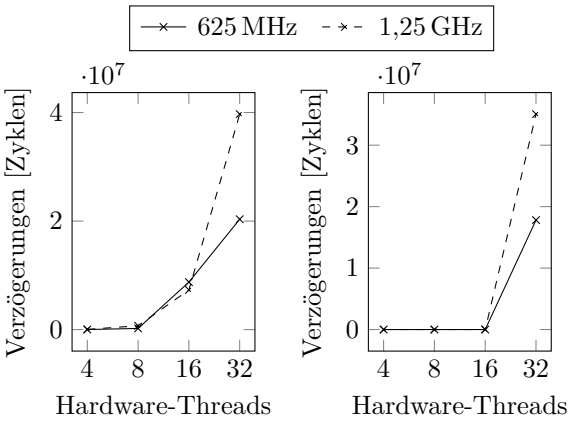


Abb. 7.18: Vier Rechenkerne sowie eine HMC-Partitionsblockgröße von 128 Byte und L2-Cacheblockgröße von 64 Byte.

lastung der Rechenpipeline im Bereich von vier bis acht Hardware-Threads signifikant gesteigert werden (Abbildung 7.18). Mit zunehmender Anzahl an Hardware-Threads fällt diese jedoch, was wiederum einhergeht mit einer Zunahme an Speicherlatenz. Da die Speicherbandbreite bis auf eine Ausnahme vollständig saturiert wurde, kann die Abnahme von Pipeline-Auslastung und Rechenleistung auf folgenden Aspekt zurückgeführt werden: Mehr Hardware-Threads stellen nahezu parallel äußerst viele Anfragen an den Ringbus des HMC, woraus eine hohe Speicherlatenz innerhalb des HMC resultiert. Da der L2-Cache daraufhin keine weiteren Anfragen an das Spei-



(a) 128 Byte (b) 64 Byte

Abb. 7.19: Vier Rechenkerne sowie eine HMC-Partitionsblockgröße von 128 Byte und L2-Cacheblockgröße von 128 Byte und 64 Byte.

chersubsystem stellen kann, häufen sich weitere Speicherzugriffe von den Hardware-Threads als Verzögerungen (engl.: *stalls*) an (Abbildung 7.19). Dies führt zu verringerter Ausführung von Hardware-Threads, da weniger Abitrierungsmöglichkeit aufgrund von vollgelaufener Warteschlangen besteht. Wie sich zeigt, kann dem eine höhere Taktfrequenz entgegenwirken, da auf sich leerende Warteschlangen des L2-Caches schneller, soll heißen mit füllen derer, agiert werden kann, womit höhere Rechenleistung generiert werden kann (Abbildung 7.18). Als Resultat können bereits wenige GPU-Rechenkerne innerhalb eines solchen PIMs eine höhere Rechenleistung erzielen, da diese weniger Interferenz innerhalb des Speichersubsystems erzeugen und somit bereits eine Saturation der Speicherbandbreite erwirken. Hinzu kommt, dass mehr parallele Hardware-Threads an einer gemeinsamen Hardware-Barriere synchronisiert werden müssen. Nach jeder Synchronisation laufen diese mit demselben Lade-/Speichermuster wieder an, weshalb nahezu zeitgleich Anfragen an das Speichersubsystem gestellt werden.

In Summe bedeutet dies im Hinblick auf die Recheneffizienz, dass vier Rechenkerne mit vier Hardware-Threads bei einer Taktfrequenz von 625 MHz 24,04 % bzw. bei 1,25 GHz 14,13 % erzielen. Wird hierzu Abschnitt 7.4.2 herangezogen, welcher für den besten Fall eine Effizienz von maximal 30 % ausweist, erscheint dies zumindest für die niedrigere Taktfrequenz respektabel.

Obgleich eine Verkleinerung der internen HMC-Partitionsblockgröße auf 64 Byte ebenfalls möglich ist, erzielt dies keine weitere Beschleunigung (Abbildung 7.20). Einzig die Speicherlatenz kann für 16 und 32 Hardware-Threads signifikant reduziert werden.

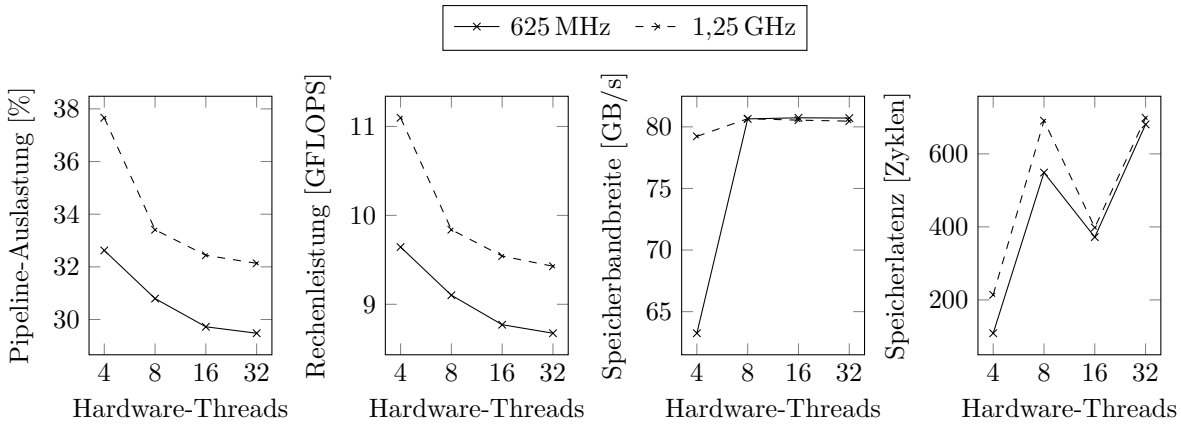


Abb. 7.20: Vier Rechenkerne sowie eine HMC-Partitionsblockgröße und L2-Cacheblockgröße von 64 Byte.

Skalierung der GPU-Rechenkerne

Mit einer Verdopplung an SLIDs halbiert sich jeweils die verfügbare Bandbreite je Rechenkern (Unterabschnitt 6.3.2). In Kombination mit Abschnitt 7.4.3 bedeutet dies, dass prinzipiell von wenig bis kaum Beschleunigung auszugehen ist, sofern weitere GPU-Rechenkerne in den PIM eingebettet werden sollen. Ausgehend davon, dass vier Hardware-Threads die meiste Fließkommarechengeschwindigkeit erzielten, wurde die Anzahl derer fest gesetzt, wohingegen die Anzahl der Rechenkerne zwischen einem und 32 variiert wurde. Wie zuvor beim Variieren der Anzahl

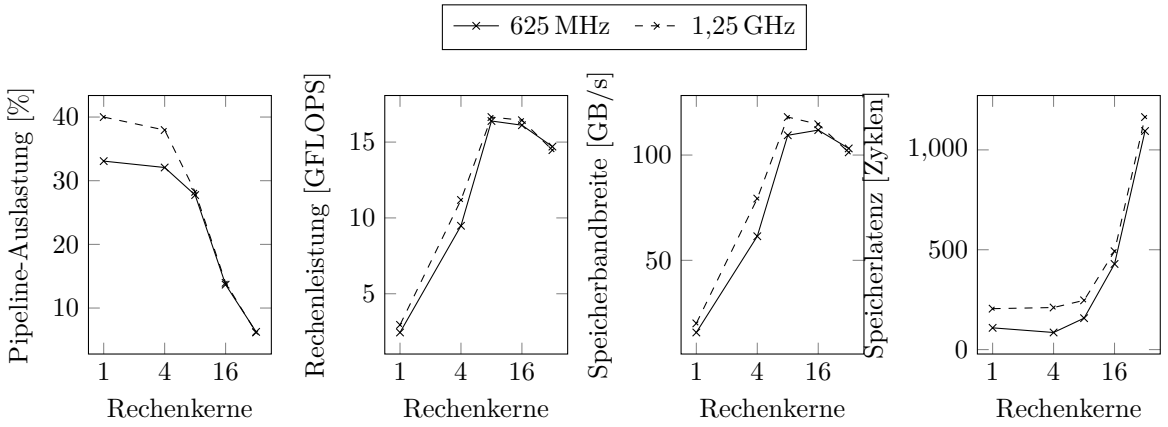


Abb. 7.21: Vier Hardware-Threads sowie eine HMC-Partitionsblockgröße von 128 Byte und L2-Cacheblockgröße von 64 Byte.

an Hardware-Threads vermindert sich die Auslastung der Verarbeitungspipeline mit Zunahme an GPU-Rechenkernen. Zeitgleich steigt die Fließkommarechenleistung, bis diese bei acht Rechenkernen ihr Maximum mit ca. 16 GFLOPS erreicht, um daraufhin wieder zu fallen. Gleiches spiegelt sich in der Speicherbandbreite wieder, welche ebenfalls bei acht Rechenkernen ihren Spitzenwert mit 115,58 GB/s bei einer Taktfrequenz von 625 MHz bzw. 127,75 GB/s bei einer Taktfrequenz von 1,25 GHz erreicht. In Angesicht dessen, dass im Falle von 625 MHz 104,97 GB/s bzw. bei 1,25 GHz 114,75 GB/s auf das Lesen entfallen, die DRAM-Speicherbandbreite maximal 320 GB/s und somit 160 GB/s für Lesen zur Verfügung stellt, kommt dies einem akzeptablen Ergebnis gleich (∼72 % Ausbeute der Lese-Speicherbandbreite). Ab acht Rechenkernen nimmt

die Rechenleistung sowie die genutzte Speicherbandbreite ab. Grund hierfür stellt einerseits das Abschnüren der Speicherbandbreite je externer Schnittstelle dar, andererseits weist die Speicherlatenz einen signifikanten Anstieg auf.

Zur Gegenprüfung, dass im Entwurfsraum tatsächlich keine weiteren Konfigurationen bestehen, die wiederum eine höhere Fließkommarechenleistung erzielen können, wurde die Limitierung auf vier Hardware-Threads bei Skalierung der Rechenkerne gelockert. Werden alle Kombinationen von 1, 4, 8, 16 und 32 Rechenkernen sowie 4, 8, 16 und 32 Hardware-Threads bei einer Taktfrequenz von 625 MHz und 1,25 GHz in Kombination mit der L2-Cacheblockgröße von 64, 128 und 256 Byte sowie HMC-Partitionsblockgröße von 32, 64, 128 und 256 Byte evaluiert, so zeigt sich, dass eine Konfiguration mit acht Rechenkernen, vier Hardware-Threads mit Taktfrequenz von 1,25 GHz einer L2-Cacheblockgröße von 64 Byte und HMC-Partitionsblockgröße von 128 Byte die größtmögliche Fließkommarechenleistung von 17,84 GFLOPS sowie die größte Lese-Speicherbandbreite von 114,75 GB/s erzeugen kann. Diese ist mit 11,15 % Fließkommarecheneffizienz zwar nicht allzu hoch, jedoch führt das Mehr an genutzter Speicherbandbreite zu höherer Fließkommarechenleistung. Steht rein die Recheneffizienz im Fokus, so ist eine Kombination aus einem Rechenkern, vier Hardware-Threads bei einer Taktfrequenz von 625 MHz sowie L2-Cacheblockgröße von 64 Byte und HMC-Partitionsblockgröße von 128 Byte mit 24,48 % am effizientesten. Jedoch wird hieraus gerade einmal eine Fließkommarechenleistung von 2,48 GFLOPS abgerufen.

7.5 Zusammenfassung

Das entwickelte Simulationsmodell des *Hybrid Memory Cube* als auch das Modell der *NYU-ZI*-ISA basierenden GPU wurden einerseits eigenständig, andererseits als integriertes PIM-Simulationswerkzeug hinsichtlich Flexibilität, Simulationsgeschwindigkeit und Funktionsumfang illustriert und evaluiert.

Durch Aktivierung des erarbeiteten Simulationstaktbaums ergibt sich eine äußerst hohe Simulationsgeschwindigkeit. Diese liegt in Teilen um einen Faktor 100 höher als existierende funktionale Ansätze wie etwa *HMC-Sim*. Wird eine exakte Speicherlatenz und Speicherbandbreite benötigt, fällt die Beschleunigung aufgrund des äußerst hohen Rechenaufwands des taktgenauen Simulators *BOBSim* geringer aus, ist jedoch noch um den Faktor zwei schneller als ohne Simulationstaktbaum. Bedingt durch die auf elementarer Datenpaketebene implementierte Logikebene, bietet das HMC-Modell eine hohe Genauigkeit, welche exemplarisch anhand eines traversierenden Pakets mit einem *RD256*-Lesebefehl nachverfolgt und vermessen wurde. Da der Simulator *BOBSim* für die Taktgenauigkeit innerhalb eines *Vaults* sorgt, ist mit der entwickelten Logikebene eine vollständig taktgenaue Simulation gegeben. Das genannte *RD256*-Beispiel nutzt externe SLIDs und interne *Links*, wodurch gleichzeitig eine Multi-HMC-Umgebung und implizit der integrierte Routing-Algorithmus demonstriert wurde. Zur Veranschaulichung wurde darüber hinaus die Möglichkeit zur Aufzeichnung der traversierenden Pakete aufgezeigt, welche zur Visualisierung von Latenz-Hotspots Anwendung finden kann. Somit resultiert, dass ein flexibles, vollumfängliches Werkzeug geschaffen wurde und vermessen werden konnte, welches für unterschiedlichste zukünftige Szenarien für HMC- sowie PIM-Studien genutzt werden kann.

Für die Analyse des GPU-Simulationsmodells sowie der PIM-Simulationsumgebung wurde eine Reverse Time Migration mit 9-Punkt-*Stencil* hergeleitet, welche eine unoptimierte arithme-

tische Intensität von 0,2333 $\frac{\text{FLOP}}{\text{Byte}}$ aufweist. Entsprechend ist diese ein idealer Repräsentant für signifikant durch Speicherbandbreite beschränkte HPC-Algorithmen. Vorab jedoch wurde die Anwendung *Rotozoom* auf dem GPU-Simulationsmodell ausgeführt, um dabei die graphische Ausgabe und die Einbettung in *SoCRocket* zu demonstrieren. Wohingegen das offizielle Verilator-Modell einen dedizierten DRAM-Speicher nutzt, besitzt das entwickelte GPU-Simulationsmodell ausschließlich eine statische Latenz zur Approximation des Hauptspeichers. Es zeigt sich, dass bei Verwendung der seismischen RTM eine Speicherlatenz von 39 Taktzyklen nahezu exakt zu einer Überschneidung führt, sodass der Simulator als annähernd taktgenau angesehen werden darf. Eine detaillierte Betrachtung weist hierbei eine kaum wahrnehmbare Abweichung im Bereich der Verarbeitungspipeline und der L1i/d-Caches auf, wohingegen bedingt durch die statische Speicherlatenz der L2-Cache sich stärker von der Hardware entfernt. Im Allgemeinen fällt jedoch die durchschnittliche Abweichung mit 6 % gering aus. Die Frage beantwortend, warum ein Simulator entwickelt werden muss, wenn bereits ein Verilator-Modell aus SystemVerilog kompiliert werden kann, zeigt die Analyse der Simulationsgeschwindigkeit auf. Hierbei skaliert das entwickelte Modell nahezu mit der Simulationsgeschwindigkeit des funktionalen Emulators, wohingegen das Verilator-Modell bei bereits kleinen RTM-Problemstellungen mehrere Größenordnungen an Zeit benötigt. Entsprechend bietet das entwickelte GPU-Simulationswerkzeug für große Problemstellungen einen signifikanten Mehrwert, bei welchen ein Ergebnis schnell mit approximierten Zeitverhalten erzielt werden soll.

Schließlich wird auf das PIM-Simulationswerkzeug, entsprechend auf das integrierte Modell beider Simulatoren, eingegangen. In Retrospektive zeigt sich dabei, dass die gewählte RTM bei Verwendung des PIM-Rechenbeschleunigers mit internem HMC-Ringbus idealisiert eine maximale Recheneffizienz von 30,0 % erzielen kann. Real fällt diese leicht auf 24,5 % ab. Da die Hälfte der realen Recheneffizienz in der aktuellen Literatur für das Höchstleistungsrechnen bereits als effizient ausgewiesen wird, weist dies eine signifikante Recheneffizienz des illustrierten PIM-Architektur aus. Insbesondere da diese noch keine Anpassungen der Instruktionen sowie Verarbeitungspipeline erhielt, existiert zugleich noch weiteres Potential. Die hierbei schnellste Kombination setzt sich aus acht GPU-Rechenkernen mit jeweils vier Hardware-Threads zusammen (insgesamt 32 Hardware-Threads). Diese erzielt eine maximale Fließkommarechenleistung von 17,84 GFLOPS bei einer maximalen Auslastung der Lese-/Speicherbandbreite von 72 % je PIM. Hierbei muss der GPGPU-Rechenbeschleuniger mit der gleichen Taktfrequenz wie HMC betrieben werden, damit die Lade-/Speicherwarteschlangen des Beschleunigers gefüllt gehalten werden können. Das Ergebnis ist hervorragend, wird in Betracht gezogen, dass hierbei noch Kommunikation auf den Inter-Quadrant-Verbindungen vonstatten geht. Wird davon ausgegangen, dass bis zu acht Speicherwürfel pro Hauptprozessor laut HMC-Spezifikation kombiniert werden können, so wäre eine solche Konfiguration in der Lage bei Ausführung des gewählten seismischen Algorithmus etwa 140 GFLOPS an realer Fließkommarechenleistung abzurufen.

Kapitel 8

Zusammenfassung und Ausblick

Speicherarchitekturen stellen das Rückgrat einer jeweiligen *Von-Neumann-Architektur* dar, limitieren jedoch aufgrund von geringer Bandbreite die zukünftige Skalierung der Rechenleistung (*Memory Wall*). Schwerfälligkeit, Ineffizienz, Unzuverlässigkeit sind Eigenschaften, welche Speicherarchitekturen zugeschrieben werden, weshalb ein Bedarf besteht den Stand der Technik zu verbessern. Hierzu stellt diese Arbeit die Technik der Re-Integration von Rechenkapazität in Speicher vor, welche unter dem Titel *Processing-In-Memory* in den 1960-er bis 1990-er Jahren in der Forschung vertieft untersucht wurde. Neue Technologien wie die *Through-Silicon Vias* (TSV), die ein effizientes dreidimensionales Stapeln erst ermöglichen, vermögen alte technische Problemstellungen zu überwinden, um neue PIM-Architekturen zu ermöglichen. Da PIM jedoch ein komplex heterogenes Rechensystem darstellen, steht derartigen neuen Forschungsansätzen ein Mangel an geeigneten Werkzeugen zur Modellierung gegenüber, weshalb sich diese Arbeit dieser Problemstellung annimmt.

Entsprechend wurde ein Simulationsmodell für dreidimensionale Speicherstapel entwickelt, welches einerseits flexibel, taktgenau und detailliert ist, andererseits äußerst schnell simulieren kann. Die Flexibilität verhilft zur standardkonformen Modellierung der *Hybrid Memory Cube*-Spezifikation 2.1, wohingegen der integrierte Simulator *BOBSim* sowie die Simulation auf elementaren Datenpaketen eine Taktgenauigkeit ermöglichen. Als Kombination bereits den Stand der Technik übertreffend, erlaubt ein Routing Mechanismus zugleich eine einzigartige Multi-HMC-Simulation, wobei ein eigens entwickelter Simulationstaktbaum für hohe Ausführungsgeschwindigkeit sorgt. Auf Grundlage einer intendierten PIM-Architektur wurde zusätzlich ein GPU-Simulationsmodell implementiert, welches zur *NYUZI*-ISA binärkompatibel, taktgenau, ebenfalls flexibel und schnell ausführend ist. Zusammengesetzt als PIM-Beschleuniger, wurden die Rechenkerne zur realitätsnahen Simulation anhand PIM in der Literatur hinsichtlich der genutzten Ressourcen angepasst. Als Anwendungsfall dient die als erste Exascale-Problemstellung identifizierte, für die Öl- und Gasindustrie wichtige Seismik, innerhalb derer die geophysikalische Bildverarbeitung einen signifikant hohen Bedarf an Rechenleistung hat, jedoch speichergebunden ist. Entsprechend wurde eine repräsentativ gewählte, seismische Reverse Time Migration als Anwendungsschwerpunkt zur Evaluierung herangezogen und auf die erarbeitete PIM-Architektur portiert und optimiert. Hierbei erzielt der PIM-Beschleuniger eine synthetische Recheneffizienz von 30,0 %, welche real leicht auf 24,5 % abfällt, was einer Verdopplung des Stands der Technik gleich kommt. Somit zeigt sich, dass das erarbeitete Werkzeug zur Modellierung von dreidimensionalen PIM-Architekturen eine tatsächliche Simulation ermöglicht und dass das intendierte

Architekturkonzept für mehr Rechenleistung sorgen kann.

8.1 Zusammenfassung

Während flüchtige Speicherarchitekturen seit Wulf und McKees Schilderung einer *Memory Wall* als Limitierung bekannt sind, verschärft sich die Skalierung der Rechengeschwindigkeit seit dem Jahr 2017, da das Mooresche Gesetz, welches besagt, dass die Transistordichte sich alle 18 Monate verdoppelt und der daraus abgeleiteten, stetigen Leistungszunahme, nicht mehr stand hält. Zugleich erscheinen neue Anwendungen mit sogenannter speicherintensiver Charakteristik, welche durch äußerst geringe Datenlokalität etablierte Cachehierarchien – bestehend aus unterschiedlichen flüchtigen Speichertechnologien – an ihre Grenzen bringen, da diese wiederum das Konzept der Datenlokalität als Grundlage ausnutzen. Um trotzdem die gewünschte Skalierung zu realisieren, bedarf es neuer Ideen für flüchtige Speicherarchitekturen, womit der Ausfall des Mooreschen Gesetzes in Kombination mit den althergebrachten *Brick Walls* kompensiert werden kann. Insbesondere hiervon betroffen sind Höchstleistungsrechensysteme, welche aktuell im Kontext der ExaFLOPS-Hürde von Industrie, Akademien und Laboren weiterentwickelt werden. Sich der Schwachstelle bewusst und aus nationalem Interesse, finanzieren daher unterschiedliche Nationen, wie etwa die USA, Europa, Japan und China im Rahmen der Thematik Exascale neuartige Rechnerarchitekturen, um dennoch eine Zunahme an Rechenleistung zu ermöglichen.

Eine tiefe Literaturanalyse offenbart, dass bereits in der Vergangenheit Anstrengungen im Bereich der Integration von Rechenkapazität in den Speicher ausgeführt wurden. Die Konzepte solcher Architekturen wurden unter der Begrifflichkeit *Processing-In-Memory* (PIM) beschrieben, welche ein Teilbereich der datennahen Rechenverarbeitung (*Near-Data Processing*, NDP) ist. Frühe PIM-Forschung konnte hierzu anhand von planarer Integration in DRAM-Speicher die technische Realisierung von speicherintegrierter Rechenkapazität illustrieren, wobei die Einbettung selbst meist neben den Bänken, hinter den Stromverstärkern (engl.: *sense amplifiers*) oder vor sowie hinter dem internen Speicherbus stattfand. Da nicht auf flüchtige Speicher begrenzt, existieren weitere Studien, welche das Konzept der Re-Integration auf die vollständige Speicherhierarchie angewandt haben, weshalb bereits Rechenkapazität in Netzwerk, magnetische Festplatten und FLASH-NAND-Speicher integriert wurde. Dabei konnte jeweils eine massive Rechenbeschleunigung aufgezeigt werden, sofern angepasste Architekturen – meist jedoch an eine Funktion gebundene Architekturen (engl.: *bounded-operand PIM operations*, BPO) – oder aber ASICs, ASIPs, CGRAs und GPU genutzt wurden, um wie im Falle von flüchtigen Speichern die ungebündelte Speicherbandbreite zu bewältigen.

Aus der planaren Integration resultierte jedoch der Wahlzwang für eine Prozesstechnologie für Logik und Speicher, weshalb zwischen großer Speicherkapazität und langsamen Recheneinheiten oder kleiner Speicherkapazität und schnellen Recheneinheiten entschieden werden musste. Gleichsam existierten weder ein ausgereiftes Programmier- noch ein tragfähiges Geschäftsmodell. Letzteres als Konsequenz dessen, dass Hersteller von flüchtigen Speichern durch die Kapazität pro Fläche entsprechend Kosten getrieben sind und damit im harten Wettbewerb Fläche für Logik nicht einräumen können. Ein Durchbruch erscheint mit der TSV-Durchkontaktierungstechnologie gelungen, können nun mehrere disjunkte Schichten übereinander gestapelt werden, sodass angesichts von Speicherarchitekturen die Bandbreite sich einfach skalieren und die notwendige Fläche verkleinern lässt. Wohingegen die ersten gestapelten Chips im Bereich der nichtflüchtigen Spei-

cher namentlich V-NAND vorzufinden sind, können auch flüchtige Speicher, wie zum Beispiel *High Bandwidth Memory* (HBM), als kommerzielle Ableger ausgemacht werden. Ein besonders interessanter Kandidat stellt *Hybrid Memory Cube* (HMC) der Firma Micron dar, welcher eine schnelle Logikebene mit mehreren langsamen Speicherebenen verbindet und zusätzlich ein abstraktes Protokoll zum Lesen und Schreiben anbietet. Obgleich nicht direkt von Micron angeboten, können in der Theorie dedizierte Rechenelemente im Sinne eines PIMs in solch eine Logikebene integriert werden. Dies würde die Kontaktstiftlimitierung vermeiden und zeitgleich signifikant mehr parallele Speicherbandbreite durch den weiten internen Bus entsprechenden Rechenelementen anbieten, wodurch *Memory Wall* und *Bandwidth Wall* gleichsam gelindert werden könnte. Zusätzlich existieren neue Programmiermodelle wie etwa das Konzept der *Heterogeneous System Architecture* (HSA) der Firma AMD, welches durch verschiedene Abstraktionsebenen explizit heterogene Rechensysteme adressiert.

Ausgehend vom PIM-Konzept wurde daher eine zu modellierende, intendierte PIM-Architektur erarbeitet, welche einerseits auf einen dreidimensionalen Speicherstapel, andererseits auf einen Rechenbeschleuniger setzt. Hierzu wurde HMC als dreidimensionale Speichertechnologie eingeplant, dessen Logikebene aufgrund von schwammiger HMC-Spezifikation mit einem Ringbus zur Kommunikation zwischen den sogenannten Quadranten und entsprechenden Rechenkernen ausgestattet wurde. Einzubettende Rechenkerne sollten daraufhin vor dem Ringbus platziert werden, sodass diese eine vollständige Speichersicht erhalten. Für den Beschleuniger wurde eine Grafikkartenarchitektur gewählt, da hierfür bereits hinreichende Programmiersprachen wie CUDA und OpenCL existieren. Zugleich sind GPU-Architekturen bedingt durch signifikante Anzahl an Hardware-Threads unempfindlich gegenüber hohe Speicherlatenzen und können zeitgleich damit hohe Rechenleistung erzielen. Da das komplexe Konstrukt rein als Beschleuniger vorgesehen war, kann ein generischer, zentraler Prozessor angedacht werden, welcher mit mehreren solcher PIM ausgestattet ist.

Zur Modellierung des komplexen PIM wurde eine weitere Literaturanalyse getätigt, jedoch existieren trotz des großen Potentials, welches von speicherintegrierten Verarbeitungsarchitekturen ausgeht, annähernd keine Simulationswerkzeuge, die eine holistische Entwurfsraumexploration erlauben. Dies resultiert im Bereich der Speicherarchitekturen aus der Nische des dreidimensionalen Stapelns, konzentrieren sich doch die meisten Modelle wie zum Beispiel *DRAMsim* und dessen Nachfolger sowie *NVmain* auf die taktgenaue Simulation von planaren Speichern. Abseits von diesen Simulatoren existieren insgesamt vier frei verfügbare HMC-Modelle, namentlich *HMC-Sim*, *(SST) VaultSimC*, *CasHMC* und *Simple HMC model*. Während die beiden erstgenannten *HMC-Sim* und *(SST) VaultSimC* rein funktionale Emulation betreiben, bieten die zuletzt genannten zwar taktgenaue Simulation, wobei *CasHMC* nicht validiert ist und *Simple HMC model* gegenüber einem nicht zu HMC standardkonformen dreidimensionalen Speicherstapel evaluiert wurde. Hinzu kommt, dass keines eine Umgebung mit multiplen HMC unterstützt, wobei bedingt durch die Verwendung eines Routingmechanismus eine derartige Fähigkeit nicht trivial nachgerüstet werden kann. Zeitgleich und inspiriert durch vergangene Forschung an planaren PIM-Architekturen, existieren ebenfalls nur wenige Modelle zur Simulation von GPGPU-Rechenbeschleuniger. Im Detail ist die Situation ähnlich überschaubar, sind doch insgesamt nur fünf Simulationsmodelle und ebenfalls fünf tatsächliche RTL-Implementierungen in der Literatur zu finden. Das namhafteste Simulationsmodell stellt *GPGPU-Sim* dar, welches eine Nvidia-Grafikkarte taktgenau simulieren kann. Ebenfalls ausgereift ist die RTL-Implementierung *MIAOW*, jedoch konzentriert sich diese auf Hochleistungsgrafikkarten, welche aufgrund von Hard-

wareressourcen keine Einbettung in die äußerst kleine Logikebene eines HMC zulässt. Daher stellt *NYUZI* eine Alternative dar, zielt dies doch auf die Einbettung in eingebetteten Systemen, wobei es sich zeitgleich an modernen MIC- und GPU-Architekturen orientiert. Beide Bereiche haben gemeinsam, dass die Simulationsmodelle nie mit Blick auf die Simulationsgeschwindigkeit entwickelt wurden, was zur Folge hat, dass an eine Einbettung von Rechenkernen in einen mit bereits 32 Vaults – unabhängig agierende, interne Speicherstapel – ausgestatteten HMC kaum zu denken ist – insbesondere dann nicht, wenn eine Simulationsumgebung mit multiplen HMC gewünscht wird.

Ein Ausweg zur Modellierung eines HMC-Speichers stellt *BOBSim* dar, welcher zwar primär zur Erstellung von *Buffer-On-Board*-Systemen angedacht und gegen tatsächliche Hardware validiert ist, jedoch mit einer angepassten Konfiguration als einzelner HMC-*Vault* genutzt werden kann. Mit entsprechender Implementierung einer Logikebene, welche vier Quadranten mit dem intendierten, auf elementarer Datenpaketebene simulierenden Ringbus und jeden Quadranten wiederum mit acht Instanzen eines *BOBSim*-basierten HMC-*Vault* vernetzt, ließ sich ein vollständiger, mit Speicherlatenz und -bandbreite ausgestatteter HMC und somit ein erster Schritt in Richtung PIM-Simulationsumgebung modellieren. In Anbetracht von insgesamt 32 Instanzen von *BOBSim* erschien eine Simulation mit multiplen HMCs nicht realisierbar, würde dies bei maximal acht HMCs zu 256 in der Simulation sequentiell zu taktenden Instanzen führen. Ein dedizierter Simulationstaktbaum wurde hierfür entworfen, welcher ausschließlich die Schnittstellen und Komponenten innerhalb eines solchen HMCs bedient, welche zu einem gewissen Zeitpunkt tatsächlich Datenpakete führen. Für noch größere Simulationsgeschwindigkeit sorgt ein Speicherbankmodell mit einer festen Latenz, welches zum Beispiel für die Softwareentwicklung zur optimalen Ausnutzung von unabhängigen *Vaults* genutzt werden kann, um nicht durch eine langsame Simulation behindert zu werden. Seitens des GPU-Simulationsmodells wurde auf die *NYUZI*-ISA zurückgegriffen. Obgleich ein Verilator-Modell existiert, welches die Hardware exakt nachbilden kann, wurde hierzu ein nahezu taktgenauer Simulator entwickelt, dessen Simulationsgeschwindigkeit auf schnelle Ausführung optimiert ist. Entsprechend wird die vollständige Verarbeitungspipeline mit jeglichen Stufen modelliert, sowie die Cachehierarchie einschließlich bis zur zweiten Ebene. Da binärkompatibel können kompilierte Anwendungen entweder auf dem Emulator, dem entwickelten Simulationsmodell oder dem Verilator-Modell ausgeführt werden. Nachfolgend wurde das entwickelte GPU-Simulationsmodell als ein jeweiliger Rechenkern in das HMC-Modell für eine ganzheitliche PIM-Simulationsumgebung integriert. Um reale Größenverhältnisse zu wahren, wurden Kennzahlen aus der Literatur – im spezifischen die SRAM-Ressourcen des IBM Active Memory Cube – auf die Beschleunigerarchitektur angewandt, wodurch die Cachehierarchie eines jeweiligen Rechenkerns signifikante Kapazitätseinschnitte hinnehmen musste. Ausgehend von maximal 25 704 Bytes SRAM je Rechenkern musste der L2-Cache auf 8 KB verkleinert werden. Auf der Annahme, dass die Integration in den Speicher für kürzere Latenzen sorgen wird, wurde der L1d-Cache eingespart, wohingegen die nun für den L2-Cache zuständige L1d-Warteschlange für mehr ausstehende Instruktionen vergrößert wurde. Obgleich die selbe Annahme für den L1i-Cache zutreffend ist, wurde dieser mit einer Kapazität von 1 KB beibehalten, um während einer jeweiligen Programmausführung kaum bis keine Interferenzen auf den L2-Cache zu generieren. Da während der Modellierung unklar blieb, wie viele Rechenkerne, welche Taktfrequenz oder gar welche Konfiguration für den L2-Cache notwendig ist, wurden jegliche Einstellungen parametrisiert, sodass ein äußerst flexibles Simulationswerkzeug entstand. Letztlich folgte eine Anpassung der Systemsoftware um eine beliebige Anzahl an Rechenkernen sowie Hardware-Threads zu un-

terstützen.

Mit Hilfe von Lade-/Speicherverlaufsaufzeichnungen des Benchmarks SPEC-CPU2006 fand eine Evaluierung des erstellten HMC-Simulationsmodells auf dessen Ausführungsgeschwindigkeit statt. Dabei konnte der eigens entwickelte Simulationstaktbaum im funktionalen Modus eine signifikante, 100-fache Beschleunigung gegenüber traditionellen Simulationsansätzen, wie etwa verfolgt von *HMC-Sim*, aufweisen. Bei Taktgenauigkeit verblieb eine Beschleunigung um den Faktor zwei, was den rechenaufwändigen 32 Instanzen des Simulators *BOBSim* zur Modellierung der HMC-*Vaults* geschuldet ist. Trotzdem ist eine schnelle Ausführung des entwickelten HMC-Simulationsmodells garantiert. Da *BOBSim* bereits gegen *Buffer-On-Board*-Hardware validiert wurde, ein HMC-*Vaults* selbst ein ähnliches Konstrukt aus einer Speicheransteuerung mit mehreren Speicherbänken darstellt, kann der modellierte HMC-*Vault* als exzellente Approximation angesehen werden. Die Logikebene stellt aufgrund schwammiger HMC-Spezifikation eine Eigenentwicklung dar, weshalb eine Validierung ausgeführt wurde. Es zeigte sich, dass der HMC-Ringbus ebenfalls taktgenau simuliert, wodurch das vollständig entwickelte HMC-Simulationsmodell für zukünftige Studien eine hervorragende Ausgangsbasis darstellt.

Für die weitere Evaluierung des GPU-Simulationsmodell sowie den integrierten PIM-Beschleuniger wurde das Teilgebiet der angewandten Geophysik, im Detail die Seismik herangezogen, da diese als erste Exascale-Problemstellung identifiziert wurde und für die Öl- und Gasindustrie von höchstem Interesse ist. Geophysikalische Bildverarbeitung, Teil der Seismik, stellt demzufolge signifikante Anforderungen an die Rechenleistung sowie im spezifischen an die Speichersubsysteme, sodass deren zukünftige Skalierung von äußerst hoher Wichtigkeit ist. Entsprechend ist die Seismik ein exzellentes Anwendungsszenario für neue dreidimensionale PIM-Beschleunigerarchitekturen, weshalb repräsentativ eine zweidimensionale Reverse Time Migration mit 9-Punkt-*Stencil* zweiter Ableitung und vierter Ordnung erarbeitet wurde. Die repräsentativ gewählte ist bedingt durch ihre arithmetische Intensität von 0,7 $\frac{\text{FLOP}}{\text{Byte}}$ durch die Speicherbandbreite beschränkt. Mithilfe einer Implementierung auf die gewählte GPU-Architektur fand eine Evaluation des mit mehreren statischen DRAM-Latenzen betriebenen GPU-Simulationsmodells, gegenüber dem Emulator und dem äußerst präzise simulierenden Verilator-Modell statt. Es zeigte sich, dass das entwickelte GPU-Modell trotz Simulation der Hardwarecharakteristik nur unwesentlich mehr Zeit benötigte als der Emulator und mit einer statischen DRAM-Latenz von 39 Zyklen nur eine maximale Abweichung von 6 % gegenüber dem Verilator-Modell aufwies. Das Verilator-Modell benötigte hingegen signifikant mehr Zeit, weshalb das GPU-Simulationsmodell bei kleiner, hinnehmbarer Abweichung für große, längliche Problemstellungen eine exzellente Approximation darstellt.

Integriert in das PIM-Simulationswerkzeug konnte das GPU-Simulationsmodell als Demonstrator zur Modellierung und Simulation einer PIM-Architektur genutzt werden. Hierzu wurden unterschiedliche Konfiguration von Rechenkernen, Hardware-Threads, L2-Cachekapazität generiert und mithilfe des selbigen Algorithmus evaluiert. Mit einer synthetischen Recheneffizienz von 30,0 % bzw. tatsächlich von 24,5 % übertraf die PIM-Konfiguration mit vier GPU-Rechenkernen und jeweils vier Hardware-Threads den Stand der Technik signifikant. Der Stand der Technik weist hiernach bereits die Hälfte der tatsächlich erreichten Effizienz für Höchstleistungsrechensysteme als hoch effizient aus, weshalb die erstellte PIM-Konfiguration mit GPU-Architektur hochprofitabel für seismisch geophysikalische Anwendungen ist. Mit verdoppelter Anzahl an GPU-Rechenkernen konnte die maximale Lese-/Speicherbandbreite von 72 % abgerufen werden, wobei insgesamt 17,9 GFLOPS erzielt wurden. Obgleich rein als Demonstrator für das PIM-

Simulationswerkzeug angedacht, illustriert die erarbeite PIM-Konfiguration zusammenfassend für speicherbeschränkten Algorithmen – wie der Seismik – eine äußerst hohe Rechengeschwindigkeit und Recheneffizienz. Dadurch stellen PIM-Architekturen einen idealen Kandidaten für zukünftige Leistungssteigerung dar, wobei das erzeugte PIM-Simulationswerkzeug hierfür eine exzellente Ausgangssituation für weiterführende Entwurfsraumexplorationen bietet.

8.2 Weiterführende Arbeiten

Trotz des Schwerpunkts auf das Simulationswerkzeug zur Modellierung von speicherintegrierten Verarbeitungsarchitekturen, wurden bereits zwei Ansätze zur Verwertung einzelner Komponenten dessen vorangetrieben.

Für den ersten Ansatz wurde das entwickelte *NYUZI*-basierte GPU-Simulationsmodell in die virtuelle Plattform *SoCRocket* integriert, um dort primär als graphische Ausgabe zu dienen. Weitere Experimente im Rahmen der Thematik *Heterogeneous System Architecture* (HSA) sind von Interesse, in welchem das Simulationsmodell als tatsächlicher Rechenbeschleuniger angewandt wird. Hauptziel wäre eine AMD-inspirierte *Accelerated Processing Unit* (APU), welche jedoch bedingt durch die frei verfügbare Hardware-IP beider Architekturen als einzige quelloffene *Open-Source*-APU vermarktet werden könnte. Zudem böte diese die Möglichkeit alle involvierten Softwarekomponenten einer HSA, etwa virtueller Instruktionssatz, Laufzeitumgebung und Arbitrierung holistisch auf Hardware/Software Co-Design zu evaluieren, um Komponenten zu identifizieren, welche durch dedizierte Hardware beschleunigt werden kann. Hierzu muss jedoch vorab eine Anpassung der SPARC-Systemsoftware sowie eine MMU-Adaption für das GPU-Simulationsmodell durchgeführt werden, wodurch SPARC-SoC und GPU die selbe Speichersicht haben und somit einen gemeinsamen Adressraum bedienen können. Nachfolgend steht eine reale Hardware-Integration an, bei welcher entweder ein AMBA-/AXI-Protokollübersetzungs-IP zu implementieren oder noch besser eine Adaptierung des AMBA-Protokolls in die GPU-Implementierung vorzunehmen ist.

Ein weiterer Ansatz ist der bereits erbrachte *Wrapper* für die Simulationswerkzeugsammlung *The Structural Simulation Toolkit* (SST). Hierzu wurde sich des SST-*Hooks* des Simulators *HMC-Sim* bedient, indem dessen API modelliert wurde, um Aufrufe auf diese in die API des innerhalb dieser Arbeit entwickelten HMC-Simulationsmodells zu übersetzen. Wohingegen nur die Funktionalität evaluiert wurde, d.h. das erfolgreiche Kompilieren mit neuer Simulator-Bibliothek sowie erste Ausführungen von SST zur Veranschaulichung von Lade-/Speicheroperationen auf den Simulator, steht eine tiefe Analyse hingegen noch aus. Grundsätzlich ist ein direkter Vergleich mit dem in SST integrierten *VaultSimC* angedacht, um die Simulationsgenauigkeit zu eruieren. Langfristig könnte der von IBM in SST eingebrachte Speichersimulator *CramSim* genutzt werden, um die Speicherpartitionen, im Detail die Bänke, innerhalb des entwickelten HMC-Simulationsmodells zu modellieren. Grund hierfür ist die primäre Ausrichtung seitens *CramSim*, welcher gleichsam vorab mit der Prämisse Simulationsgeschwindigkeit entwickelt und zusätzlich jedoch gegen tatsächliche Hardware validiert wurde. Dadurch wäre eine Loslösung von dem rechenaufwändigen *BOBSim* möglich, welcher genutzt wurde, da die Alternative *CramSim* erst im Jahr 2017 zur Verfügung stand.

Obgleich bereits die Kernmerkmale der Rechenleistung und Recheneffizienz des mit GPU-Rechenkernen ausgestatteten PIM-Beschleunigers anhand verschiedener Konfigurationen unter-

sucht wurden, lassen sich noch Fragen hinsichtlich des Energieverbrauchs und der Thermik formulieren. Ersterer könnte anhand von PIM-Kenndaten aus dem AMC-Forschungsprojekt extrahiert werden, um wiederum Rückschlüsse auf die thermische Belastung zu ziehen. Diese wird seitens der Aufbau- und Verbindungstechnik in der Literatur als Schlüsselherausforderung für PIM-Architekturen erkannt, wird jedoch bei Studien häufig nicht näher untersucht. Insbesondere im Hinblick auf den HMC-Aufbau mit der Logikebene in der untersten Lage, ist solch Evaluierung von höchstem Interesse, würden doch die eingebetteten Rechenelemente die darüber liegenden Speicherbänke signifikant erhitzen. Entsprechend könnten Zuverlässigkeit und Belastbarkeit unter Umständen nicht mehr gewahrt werden.

Zusätzlich fehlt eine Untersuchung geeigneter Programmiersprachen sowie Programmiermodelle. Ein erster Ansatz wäre eine Sprache, welche leicht Datenstrukturen auf physikalische Adressverwürfelung abbilden kann um dadurch Lokalität überhaupt ausdrückbar zu machen. Klar ist, dass dies nicht in Hochsprachen verfasste Programmtexte adressiert, sondern hardwarenah abzubildende Algorithmen, welche eine ebenfalls hardwarenahe Sprache zur performanten Abbildung auf derartige Hardware benötigen. Ein Sprachbeispiel aus dem Bereich der *Partitioned Global Address Space* (PGAS) ist die Sprache *Legion*, welche für verteilte, heterogene Rechensysteme eine datenzentrische, hochparallele Programmierung erlaubt [Bauer et al., 2012]. *Legion* selbst baut auf der Meta-Programmiersprache *Terra* auf, welche wiederum wie eine typische C/C++-Programmierung ein manuelles Speichermanagement erwartet [DeVito et al., 2013].

8.3 Fazit

Speicherintegrierte Rechenverarbeitung in Kombination mit dreidimensionaler Stapelung offeriert aufgrund von kürzeren und einem Vielfachen an Verbindungen, sowie einer engeren Integration einen signifikanten Rechenleistungsgewinn. Allerdings ist die Rechenarchitektur sowie der interne Aufbau des dreidimensionalen Speicherstapels ausschlaggebend für den Erfolg des Konzepts. Obgleich in unterschiedlichen Studien bereits ausgiebig evaluiert, steht eine Kommerzialisierung mit entsprechendem Erfolg noch aus. Jedoch machen dreidimensional flüchtige sowie nichtflüchtige Speicherstapel, wie etwa High Bandwidth Memory und V-NAND Mut, dass in der Zukunft ggfs. auch dreidimensionale PIM-Beschleunigerarchitekturen als Produkt erwerbbar sein werden. Exemplarisch steht hierzu der nahezu marktreife *Hybrid Memory Cube*, welcher bereits PIM-ähnliche atomare Instruktionen innerhalb des Speichers ausführt, wodurch dies von Verarbeitungsarchitekturen delegiert werden kann. Es mag deutlich radikalere Ansätze wie das Quantenrechnen oder die neuromorphen Architekturen geben, welche die Welt der IT vollständig auf den Kopf stellen und dabei wie im Falle des Quantenrechnens gegebene Limitierungen wie etwa die *Memory Wall* vollständig abschaffen werden (Unterabschnitt 3.3.2). Demgegenüber steht historisch gewachsene Software, welche womöglich kaum bis gar nicht durch solch neuartige Architekturen verarbeitet werden kann. Entsprechend stellen dreidimensionale PIM-Beschleunigerarchitekturen eine exzellente Alternative dar, um auch in näherer Zukunft für eine Skalierung von Rechenleistung zu sorgen. Hierzu vermag diese Arbeit zukünftigen Entwurfsraumexplorationen ein Simulationswerkzeug zur Verfügung zu stellen, welches vollumfänglich und hoch flexibel einen dreidimensionalen Speicherstapel inklusive PIM-Integrationsmöglichkeit modellieren und simulieren kann.

Anhang A

Grundlagen

A.1 Beweis des Entwicklungssatz von Taylor

Beweis. Sei $T_n(x,c) = \sum_{k=0}^{n} \frac{f^{(k)}(c)}{k!}(x-c)^k$ das Taylorpolynom, sei $g(x) = f(x) - T_n(x)$ die Differenz zwischen f und dem Taylorpolynom, dann ist

$$\begin{aligned} g(c) &= f(c) - T_n(c,c) \\ &= f(c) - f(c) = 0 \\ g'(c) &= f'(c) - T_n(c,c) \\ &= f'(c) - f'(c) = 0 \\ &\vdots \\ g^{(n)}(c) &= f^{(n)}(c) - T_n(c,c) \\ &= f^{(n)}(c) - f^{(n)}(c) = 0 \end{aligned}$$

Wird der 2. Mittelwertsatz auf $\frac{g(x)}{(x-c)^{n+1}}$ angewandt und sei ein c_1 zwischen c und x gegeben:

$$\begin{aligned} \frac{g(x)}{(x-c)^{n+1}} &= \frac{g(x)-g(c)}{(x-c)^{n+1}-(c-c)^{n+1}} \\ &= \frac{g(x)-0}{(x-c)^{n+1}-0} \\ &= \frac{g'(c_1)}{(n+1)(c_1+c)^n} && \text{2. MWS} \\ &= \frac{g''(c_2)}{(n+1)n(c_2+c)^{n-1}} && \text{Induktion} \\ &\vdots \\ &= \frac{g^{(n)}(c_n)}{(n+1)!(c_n+c)} \\ &= \frac{g^{(n+1)}(c_{n+1})}{(n+1)!} \end{aligned}$$

Mit $g(x) = f(x) - T_n(x, c)$ folgt:

$$f(x) - T_n(x, c) = \frac{(x-c)^{n+1}}{(n+1)!} g^{(n+1)}(c_n + 1)$$

Unter Beachtung von $g^{(k)}(c) = f^{(k)}(c) = 0, \forall k : 0 \leq k \leq n$ ergibt sich:

$$f(x) = T_n(x, c) + \frac{(x-c)^{n+1}}{(n+1)!} f^{(n+1)}(c_{n+1}), \forall c_{n+1} : c \leq c_{n+1} \leq x$$

□

A.2 Herleitung der Reverse Time Migration

Ausgehend von der n-dimensional, akustisch homogenen Wellengleichung (Unterabschnitt 2.2.4):

$$\Delta p - \frac{1}{c^2} \frac{\partial^2 p}{\partial t^2} = 0$$
$$\sum_{k=1}^{n} \frac{\partial^2 p}{\partial x_k^2} - \frac{1}{c^2} \frac{\partial^2 p}{\partial t^2} = 0$$

Aufgrund von Unterabschnitt 2.2.2 kann $\frac{1}{c^2} \frac{\partial^2 p}{\partial t^2}$ eindimensional vereinfacht werden, um daraufhin partiell vorwärts und rückwärts abgeleitet zu werden. Beachte: Δt stellt das diskrete Zeitintervall dar.

$$p(t + \Delta t) = p(t) + \Delta t p'(t) + \frac{\Delta t^2}{2} p''(t) + O(h^3)$$
$$p(t - \Delta t) = p(t) - \Delta t p'(t) + \frac{\Delta t^2}{2} p''(t) + O(h^3)$$

Beide Gleichungen zusammenaddiert ergibt:

$$p(t + \Delta t) + p(t - \Delta t) = 2p(t) + \Delta t^2 p''(t) + O(h^3)$$
$$p''(t) = \frac{1}{\Delta t^2}[p(t - \Delta t) + p(t + \Delta t) - 2p(t)] + O(h)$$
$$\frac{\partial^2 p}{\partial t^2} = \frac{1}{\Delta t^2}[p(t - \Delta t) + p(t + \Delta t) - 2p(t)]$$

Eingesetzt in die n-dimensional, akustisch homogene Wellengleichung:

$$\sum_{k=1}^{n} \frac{\partial^2 p}{\partial x_k^2} - \frac{1}{c^2 \Delta t^2}[p(x_1, \ldots, x_n, t - \Delta t) + p(x_1, \ldots, x_n, t + \Delta t) - 2p(x_1, \ldots, x_n, t)] = 0$$

Vorwärtsgerichtete Reverse Time Migration:

$$p(x_1, \ldots, x_n, t + \Delta t) = 2p(x_1, \ldots, x_n, t) - p(x_1, \ldots, x_n, t - \Delta t) + c^2 \Delta t^2 \sum_{k=1}^{n} \frac{\partial^2 p}{\partial x_k^2}$$

... mit Laplace-Operator auf p

$$p(x_1, \ldots, x_n, t + \Delta t) = 2p(x_1, \ldots, x_n, t) - p(x_1, \ldots, x_n, t - \Delta t) + c^2 \Delta t^2 \Delta p$$

Rückwärts-gerichtete Reverse Time Migration:

$$p(x_1, \ldots, x_n, t - \Delta t) = 2p(x_1, \ldots, x_n, t) - p(x_1, \ldots, x_n, t + \Delta t) + c^2 \Delta t^2 \sum_{k=1}^{n} \frac{\partial^2 p}{\partial x_k^2}$$

... mit Laplace-Operator auf p

$$p(x_1, \ldots, x_n, t - \Delta t) = 2p(x_1, \ldots, x_n, t) - p(x_1, \ldots, x_n, t + \Delta t) + c^2 \Delta t^2 \Delta p$$

Anhang B

Modellierung

B.1 Herleitung des Laplace-Operator-basierten 9-Punkt-*Stencil*

Für den in dieser Arbeit verwendeten, seismischen Algorithmus wurde auf einen Laplace Kreuz-*Stencil* zurückgegriffen, welcher eine vierte Ordnung besitzt und bedingt durch Laplace eine zweite Ableitung (Abbildung 7.4).

Sei $f(\vec{x}\,)$ eine zweifach differenzierbare Funktion mit Entwicklungspunkt $\vec{x} = \begin{pmatrix} x+h \\ z \end{pmatrix}$, so gilt:

$$\begin{aligned}
f(\vec{x}\,) &= \sum_{|\alpha|<5} \frac{1}{\alpha!} \cdot D^{\alpha} f(\vec{c})(\vec{x}-\vec{c})^{\alpha} + O((\vec{x}-\vec{c})^6) \\
&= \frac{1}{0!} \cdot D^0 f(\vec{c})(\vec{x}-\vec{c})^0 + \frac{1}{1!} \cdot D^1 f(\vec{c})(\vec{x}-\vec{c})^1 + \frac{1}{2!} \cdot D^2 f(\vec{c})(\vec{x}-\vec{c})^2 \\
&\quad \frac{1}{3!} \cdot D^3 f(\vec{c})(\vec{x}-\vec{c})^3 + \frac{1}{4!} \cdot D^4 f(\vec{c})(\vec{x}-\vec{c})^4 + \frac{1}{5!} \cdot D^5 f(\vec{c})(\vec{x}-\vec{c})^5 \\
&= f(\vec{c})(\vec{x}\,) + Df(\vec{c})(\vec{x}-\vec{c}) + \frac{1}{2} \cdot D^2 f(\vec{c})(\vec{x}-\vec{c})^2 \\
&\quad + \frac{1}{6} \cdot D^3 f(\vec{c})(\vec{x}-\vec{c})^3 + \frac{1}{24} \cdot D^4 f(\vec{c})(\vec{x}-\vec{c})^4 + \frac{1}{120} \cdot D^5 f(\vec{c})(\vec{x}-\vec{c})^5
\end{aligned}$$

Nachfolgende Terme wurden ab der vierten Ordnung gekürzt, da die Ableitungen nach z sich ohnehin herauskürzen.

$$\begin{aligned}
f(x+h,z) &= f(x,z) + f_x(x,z)(x+h-x) + f_z(x,z)(z-z) \\
&\quad + \frac{1}{2}[h^2 \dot{f}_{xx}(x,z) + 2 \cdot 0 \cdot f_{xz}(x,z) + 0 \cdot f_{zz}(x,z)] \\
&\quad + \frac{1}{6}[h^3 \dot{f}_{xxx}(x,z) + 3 \cdot 0 \cdot f_{xxz}(x,z) + 3 \cdot 0 \cdot f_{xzz}(x,z) + 0 \cdot f_{zzz}(x,z)] \\
&\quad + \frac{h^4}{24} \dot{f}_{xxxx}(x,z) + \frac{h^5}{120} \dot{f}_{xxxxx}(x,z) + O(h^6) \\
f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2} f^{(2)}(x) + \frac{h^3}{6} f^{(3)}(x) + \frac{h^4}{24} f^{(4)}(x) + \frac{h^5}{120} f^{(5)}(x) + O(h^6)
\end{aligned}$$

Wie gezeigt, bildet die multidimensionale Entwicklung nach Taylor für den zweidimensionalen Fall keine gemischten Ableitungen. Dadurch kann mit einfacher, eindimensionaler Entwicklung

und anschließend mit Addition der $f^{(2)}(x,z)$ Differenzen fortgefahren werden:

$$f(x-2h) = f(x) - 2hf'(x) + 2h^2 f^{(2)}(x) - \frac{4h^3}{3} f^{(3)}(x) + \frac{2h^4}{3} f^{(4)}(x) - \frac{4h^5}{15} f^{(5)}(x) + O(h^6) \tag{B.1}$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2} f^{(2)}(x) - \frac{h^3}{6} f^{(3)}(x) + \frac{h^4}{24} f^{(4)}(x) - \frac{h^5}{120} f^{(5)}(x) + O(h^6) \tag{B.2}$$

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f^{(2)}(x) + \frac{h^3}{6} f^{(3)}(x) + \frac{h^4}{24} f^{(4)}(x) + \frac{h^5}{120} f^{(5)}(x) + O(h^6) \tag{B.3}$$

$$f(x+2h) = f(x) + 2hf'(x) + 2h^2 f^{(2)}(x) + \frac{4h^3}{3} f^{(3)}(x) + \frac{2h^4}{3} f^{(4)}(x) + \frac{4h^5}{15} f^{(5)}(x) + O(h^6) \tag{B.4}$$

Ergebnis von (B.1) + (B.4) und (B.2) + (B.3):

$$\begin{aligned} f(x-2h) + f(x+2h) &= 2f(x) + 4h^2 f^{(2)}(x) + \frac{4h^3}{3} f^{(4)}(x) + O(h^6) \\ f(x-h) + f(x+h) &= 2f(x) + h^2 f^{(2)}(x) + \frac{h^4}{12} f^{(4)}(x) + O(h^6) \end{aligned} \tag{B.5}$$

Letzter Schritt: (B.5) · 16 - (B.1)

$$\begin{aligned} 16(f(x-h) + f(x+h)) - (f(x-2h) + f(x+2h)) &= 30f(x) + 12h^2 f^{(2)}(x) + O(h^6) \\ f^{(2)}(x) &= \frac{1}{12h^2}[-f(x-2h) + 16f(x-h) - 30f(x) + 16f(x+h) - f(x+2h)] + O(h^4) \end{aligned}$$

Eindimensionales Resultat in *Stencil*-Notation:

$$f''(x) \approx \frac{1}{12h^2} \begin{bmatrix} -1 & 16 & -30 & 16 & -1 \end{bmatrix} f$$

Da die erstellte Gleichung sowohl für x als auch z-Richtung gilt und das zu verwendende Gitter regulär ist, können beide zusammenaddiert werden:

$$\begin{aligned} \Delta f :&= f_{xx} + f_{zz} \\ &= \frac{1}{12h^2}[-f(x-2h,z) + 16f(x-h,z) - 30f(x,z) + 16f(x+h,z) - f(x+2h,z)] \\ &\quad + \frac{1}{12h^2}[-f(x,z-2h) + 16f(x,z-h) - 30f(x,z) + 16f(x,z+h) - f(x,z+2h)] \\ &= \frac{1}{12h^2}[-f(x-2h,z) - f(x,z-2h) - f(x+2h,z) - f(x,z+2h) \\ &\quad + 16(f(x-h,z) + f(x,z-h) + f(x+h,z) + f(x,z+h)) \\ &\quad - 60f(x,z)] \end{aligned}$$

Zweidimensionales Resultat in *Stencil*-Notation:

$$\Delta f = \frac{1}{12h^2} \begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 16 & 0 & 0 \\ -1 & 16 & -60 & 16 & -1 \\ 0 & 0 & 16 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix} f$$

B.2 Optimierte Reverse Time Migration mit 9-Punkt-*Stencil* zweiter Ableitung und vierter Ordnung

Auflistung B.1: Optimierte Implementierung des gewählten Reverse Time Migration Algorithmus mit 9-Punkt-*Stencil* für eine NYUZI-ISA.

```
1   unsigned WIDTHdiv16 = WIDTH >> 4;
2   z_bgn *= WIDTHdiv16;
3   z_end *= WIDTHdiv16;
4   unsigned t, offset, z_width = z_end - z_bgn;
5   for( t = 0; t < TIMESTEPS; t++ ) {
6     if( set_pulse )
7       lAPF[ pulse >> 4 ][ pulse & 0xf ] = pulseVec[ t ];
8
9     for( offset = z_bgn + x_bgn; offset < z_bgn + x_end; offset++ )
10    { // Vorabsequenz
11      vecf16_t *v16p_a = &lAPF[offset - 2 * WIDTHdiv16];
12      vecf16_t v16_u2 = *v16p_a;
13      vecf16_t v16_u1 = *(v16p_a += WIDTHdiv16);
14      vecf16_t v16_m = *(v16p_a += WIDTHdiv16);
15      vecf16_t v16_a1 = *(v16p_a += WIDTHdiv16);
16
17      vecf16_t *v16p_v = &v16_VEL[offset];
18      vecf16_t * v16p_n = &v16_NPPF[offset];
19      while(v16p_n < &v16_NPPF[z_width + offset])
20      { // Stencil Start
21        vecf16_t* m16_APF = v16p_a - WIDTHdiv16;
22        vecf16_t v16_a2 = *(v16p_a += WIDTHdiv16);
23        vecf16_t v16_n = *v16p_n;
24        vecf16_t v16_v = *v16p_v;
25
26        vecf16_t v16_l1 = __builtin_nyuzi_shufflef(v16_m, const_shf_r);
27        vecf16_t v16_r1 = __builtin_nyuzi_shufflef(v16_m, const_shf_l);
28        v16_l1[0] = *(((float*)m16_APF) - 1);
29        v16_r1[16-1] = *(((float*)m16_APF) + 16);
30
31        vecf16_t v16_l2 = __builtin_nyuzi_shufflef(v16_l1, const_shf_r);
32        vecf16_t v16_r2 = __builtin_nyuzi_shufflef(v16_r1, const_shf_l);
33        v16_l2[0] = *(((float*)m16_APF) - 2);
34        v16_r2[16-1] = *(((float*)m16_APF) + 16 + 1);
35
36        (*v16p_n) = const_two * v16_m - v16_n + v16_v
37                    * (const_sixteen * (v16_l1 + v16_r1 + v16_u1 +
                        v16_a1)
38                       - (v16_l2 + v16_r2 + v16_u2 + v16_a2)
39                       - const_sixty * v16_m );
40
41        v16_u2 = v16_u1;
42        v16_u1 = v16_m;
43        v16_m = v16_a1;
```

```
44          v16_a1 = v16_a2;
45          v16p_n += WIDTHdiv16;
46          v16p_v += WIDTHdiv16;
47        } // Stencil Schluss
48      }
49
50      lTMP = v16_NPPF;
51      v16_NPPF = lAPF;
52      lAPF = lTMP;
53
54      // BARRIER
55    }
```

Die Reverse Time Migration mit 9-Punkt-*Stencil* zweiter Ableitung, vierter Ordnung hat bei Verwendung der NYUZI-ISA eine arithmetische Intensität von 0,56 $\frac{\text{FLOPs}}{\text{Byte}}$ ($= \frac{16 \cdot 14\,\text{FLOPS}}{(16 \cdot 5_{ld} + 4_{ld} + 16 \cdot 1_{st}) \cdot 4\,\text{Byte}}$). Wird die innere Schleife ausgerollt und werden Register vorab geladen, sodass jeweils vier Vektorregister bei jeder neuen spaltenorientierten Iteration wiederbenutzt werden, so verbessert sich die arithmetische Intensität auf 0,82 $\frac{\text{FLOPs}}{\text{Byte}}$ ($= \frac{16 \cdot 14\,\text{FLOPS}}{(16 \cdot 3_{ld} + 4_{ld} + 16 \cdot 1_{st}) \cdot 4\,\text{Byte}}$; Quellcode: Aufl. B.1, Disassemblierter Kern: Aufl. B.2). Insgesamt benötigt eine derartige Implementierung 39 Befehle, wobei 156 Byte an Instruktionen notwendig sind. Acht dieser 39 Operationen sind dabei Lade-/Speicherbefehle, wohingegen 14 Fließkommaoperationen darstellen. Entsprechend sind nur 56,41 % der Befehle tatsächlich dem Algorithmus geschuldet, wohingegen die restlichen für Adressberechnung oder Kontrollfluß zuständig sind. Wird dieser Prozentsatz mit der hochoptimierten AI kombiniert, so ist eine tatsächliche AI_{real} von maximal 0,47 $\frac{\text{FLOPs}}{\text{Byte}}$ erwartbar. Wird die innerste *Stencil*-Implementierung in die vollständige Berechnung mit den äußeren Schleifen eingebettet, so fallen knapp 498 Byte an. Mit *main()*-Prozedur und Hardware-Barriere sind es gar 1784 Byte.

Auflistung B.2: Disassemblierte Maschinensprache des hochoptimierten RTM-*Stencil*.

```
1       20 01 f3 d0         move v9, v6
2       c0 00 f4 d0         move v6, v8
3       00 81 f2 d0         move v8, v5
4       ae 00 09 ae         load_v v5, 576(s14)
5       6b 01 00 ae         load_v v11, (s11)
6       46 01 22 d2         mul_f v10, v6, v4
7       8d 01 00 ae         load_v v12, (s13)
8       c6 81 d1 d0         shuffle v14, v6, v3
9       ee f1 f6 a9         load_32 s15, -580(s14)
10      a8 81 04 d2         add_f v13, v8, v9
11      ad 01 09 05         add_i s13, s13, 576
12      ea 80 13 d2         sub_f v7, v10, v7
13      46 01 d1 d0         shuffle v10, v6, v2
14      40 95 f7 c8         move_mask v10, s5, s15
15      ee 01 f8 a9         load_32 s15, -512(s14)
16      0e e2 f6 a9         load_32 s16, -584(s14)
17      ea 01 d1 d0         shuffle v15, v10, v2
18      4d 01 05 d2         add_f v10, v13, v10
19      c0 89 f7 c8         move_mask v14, s2, s15
20      ee 11 f8 a9         load_32 s15, -508(s14)
21      e0 15 f8 c8         move_mask v15, s5, s16
22      e7 80 12 d2         sub_f v7, v7, v5
23      ce 01 09 05         add_i s14, s14, 576
24      ae 81 d1 d0         shuffle v13, v14, v3
25      a0 89 f7 c8         move_mask v13, s2, s15
26      4a 01 07 d2         add_f v10, v10, v14
27      e7 80 17 d2         sub_f v7, v7, v15
28      4a 81 20 d2         mul_f v10, v10, v1
29      e7 80 16 d2         sub_f v7, v7, v13
30      a6 01 20 d2         mul_f v13, v6, v0
31      e7 00 05 d2         add_f v7, v7, v10
32      4d 81 15 d2         sub_f v10, v13, v11
33      e7 00 26 d2         mul_f v7, v7, v12
34      ea 80 03 d2         add_f v7, v10, v7
35      eb 00 00 8e         store_v v7, (s11)
36      6b 01 09 05         add_i s11, s11, 576
37      e0 80 f4 d0         move v7, v9
38      eb 01 86 c1         cmplt_u s15, s11, s12
39      4f fb ff f5         bnz s15, -152
```

B.3 Treiberprogramm zum Einlesen der SPEC-CPU2006-Benchmarkanwendungen

Auflistung B.3: Quellcode des Treiberprogramm, welches alle vier HMC-Schnittstellen mit dem Lade-/Speicherverlauf eines jeweiligen SPEC-CPU2006-Benchmarks betreibt (Aufl. 6.1).

```
1  void clkHmcSim(hmc_sim *sim,
2                 hmc_notify* slidnotify)
3  {
4    char retpacket[17*FLIT_WIDTH / (sizeof(char)*8)];
5    sim->clock();
6    unsigned mask = slidnotify->get_notification();
7    if(mask)
8      for(unsigned i = 0; i < 4; i++)
9        if(mask & (0x1 << i))
10         sim->hmc_recv_pkt(i, retpacket);
11 }
12
13 int main(int argc, char* argv[])
14 {
15   std::ifstream myfile (argv[1]);
16   if (!myfile.is_open()) {
17     std::cerr << "Unable␣to␣open␣file" << std::endl;
18     return -1;
19   }
20
21   hmc_sim sim(1, 4, 4, 4, HMCSIM_FULL_LINK_WIDTH, HMCSIM_BR30);
22   hmc_notify* slidnotify;
23   for(unsigned slidId = 0; slidId < 4; slidId++) {
24     slidnotify = sim.hmc_define_slid(slidId, 0, slidId /* linkId */,
25                                      HMCSIM_FULL_LINK_WIDTH, HMCSIM_BR30
                                           );
26     if (slidnotify == nullptr) {
27       std::cerr << "slid␣setup␣was␣not␣successful" << std::endl;
28       return -1;
29     }
30   }
31
32   unsigned line_count = std::count(std::istreambuf_iterator<char>(myfile
       ),
33                                    std::istreambuf_iterator<char>(), '\n
                                         ');
34   myfile.clear();
35   myfile.seekg(0, std::ios::beg);
36
37   char packet[(17*FLIT_WIDTH) / (sizeof(char)*8)];
38   unsigned slidId = 0;
39   std::string line;
40   while ( getline (myfile,line) ) {
41     std::stringstream ss(line);
```

```
42      uint64_t buf;
43      std::vector<uint64_t> tokens;
44      while (ss >> buf)
45        tokens.push_back(buf);
46
47      for(unsigned i = 0; i < (unsigned)tokens[0]; i++) // waitcyles
48        clkHmcSim(&sim, slidnotify);
49
50      if(tokens.size() == 3) { // contains also a write
51        sim.hmc_encode_pkt(0, tokens[2], 0, WR64, packet);
52        sim.hmc_send_pkt(slidId, packet);
53        if(++slidId >= 4)
54          slidId = 0;
55
56        clkHmcSim(&sim, slidnotify);
57      }
58
59      sim.hmc_encode_pkt(0, tokens[1], 0, RD64, packet);
60      unsigned tmp = 4;
61      while( ! sim.hmc_send_pkt(slidId, packet) ) {
62        if(++slidId >= 4)
63          slidId = 0;
64        tmp--;
65        if(!tmp) {
66          clkHmcSim(&sim, slidnotify);
67          tmp = 4;
68        }
69      }
70    }
71
72    myfile.close();
73    return 0;
74  }
```

Literaturverzeichnis

[Agerwala, 2014] T. Agerwala, “Data centric systems: The next paradigm in computing,” *Parallel Processing (ICPP), 2014 43rd International Conference on*, Sept 2014.

[Ahn et al., 2015] J. Ahn, S. Yoo, O. Mutlu, und K. Choi, “Pim-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture,” in *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, Serie ISCA ’15. New York, NY, USA: ACM, 2015, S. 336–348. [Online]. Available: http://doi.acm.org/10.1145/2749469.2750385

[Aimoto et al., 1996] Y. Aimoto, T. Kimura, Y. Yabe, H. Heiuchi, Y. Nakazawa, M. Motomura, T. Koga, Y. Fujita, M. Hamada, T. Tanigawa, H. Nobusawa, und K. Koyama, “A 7.68 gips 3.84 gb/s 1w parallel image processing ram integrating a 16 mb dram and 128 processors,” in *Solid-State Circuits Conference, 1996. Digest of Technical Papers. 42nd ISSCC., 1996 IEEE International*, Feb 1996, S. 372–373.

[Alakarhu und Niittylahti, 2002] J. Alakarhu und J. Niittylahti, “Dram simulator for design and analysis of digital systems.” *Microprocessors and Microsystems*, Vol. 26, Nr. 4, S. 189–198, 2002.

[Aly et al., 2015] M. M. S. Aly, M. Gao, G. Hills, C.-S. Lee, G. Pitner, M. M. Shulaker, T. F. Wu, M. Asheghi, J. Bokor, F. Franchetti, K. E. Goodson, C. Kozyrakis, I. Markov, K. Olukotun, L. Pileggi, E. Pop, J. Rabaey, C. Re, H. S. P. Wong, und S. Mitra, “Energy-efficient abundant-data computing: The n3xt 1,000x,” *Computer*, Vol. 48, Nr. 12, S. 24–33, Dec 2015.

[AMD, 2015] AMD, “The Road to the AMD "Fiji"GPU - Featuring Die Stacking and HBM Technology,” http://www.semicontaiwan.org/zh/sites/semicontaiwan.org/files/data15/docs/3_semicont_2015_-_black.pdf, September 2015, eCTC 2016.

[Amdahl, 1967] G. M. Amdahl, “Validity of the single processor approach to achieving large scale computing capabilities,” in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, Serie AFIPS ’67 (Spring). New York, NY, USA: ACM, 1967, S. 483–485. [Online]. Available: http://doi.acm.org/10.1145/1465482.1465560

[Andryc et al., 2013] K. Andryc, M. Merchant, und R. Tessier, “Flexgrip: A soft gpgpu for fpgas,” in *2013 International Conference on Field-Programmable Technology, FPT 2013, Kyoto, Japan, December 9-11, 2013*, 2013, S. 230–237. [Online]. Available: http://dx.doi.org/10.1109/FPT.2013.6718358

[Azarkhish et al., 2015] E. Azarkhish, D. Rossi, I. Loi, und L. Benini, "High performance axi-4.0 based interconnect for extensible smart memory cubes," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, S. 1317–1322.

[Azarkhish et al., 2016] E. Azarkhish, D. Rossi, I. Loi, und L. Benini, "Design and evaluation of a processing-in-memory architecture for the smart memory cube," in *Proceedings of the 29th International Conference on Architecture of Computing Systems – ARCS 2016 - Volume 9637*. New York, NY, USA: Springer-Verlag New York, Inc., 2016, S. 19–31. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-30695-7_2

[Bailey, 1997] D. H. Bailey, "Little's Law and High Performance Computing," *RNR Technical Report RNR-XX-XXX*, 1997.

[Bakhoda et al., 2009] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, und T. M. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," in *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, April 2009, S. 163–174.

[Balasubramanian et al., 2015] R. Balasubramanian, V. Gangadhar, Z. Guo, C.-H. Ho, C. Joseph, J. Menon, M. P. Drumond, R. Paul, S. Prasad, P. Valathol, und K. Sankaralingam, "Enabling gpgpu low-level hardware explorations with miaow - an open source rtl implementation of a gpgpu," *ACM Transactions on Architecture and Code Optimization*, 2015.

[Balasubramonian et al., 2014] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, und S. Swanson, "Near-data processing: Insights from a micro-46 workshop," *Micro, IEEE*, Vol. 34, Nr. 4, S. 36–42, July 2014. [Online]. Available: http://dblp.org/db/journals/micro/micro34.html#BalasubramonianCMMMNS14

[Bauer et al., 2012] M. Bauer, S. Treichler, E. Slaughter, und A. Aiken, "Legion: Expressing locality and independence with logical regions," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, Serie SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, S. 66–1. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389086

[Baumeister et al., 2015] P. F. Baumeister, H. Boettiger, J. R. Brunheroto, T. Hater, T. Maurer, A. Nobile, und D. Pleiter, *High Performance Computing: 30th International Conference, ISC High Performance 2015, Frankfurt, Germany, July 12-16, 2015, Proceedings*. Springer International Publishing, 2015, ch. Accelerating LBM and LQCD Application Kernels by In-Memory Processing, S. 96–112.

[Baysal et al., 1983] E. Baysal, D. D. Kosloff, und J. W. C. Sherwood, "Reverse time migration," *Geophysics*, Vol. 48, Nr. 11, S. 1514–1524, 1983. [Online]. Available: http://geophysics.geoscienceworld.org/content/48/11/1514

[Benton, 2017] B. Benton, "CCIX, GEN-Z, OpenCAPI: OVERVIEW & COMPARISON," in *13th ANNUAL WORKSHOP 2017*. AMD, 2017.

[Bergman et al., 2008] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards,

A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, K. Yelick, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Keckler, D. Klein, P. Kogge, R. S. Williams, und K. Yelick, "ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems Peter Kogge, Editor & Study Lead," http://www.cse.nd.edu/Reports/2008/TR-2008-13.pdf, Jan 2008, TR-2008-13.

[Binkert et al., 2011] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, und D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, Vol. 39, Nr. 2, S. 1–7, aug 2011. [Online]. Available: http://doi.acm.org/10.1145/2024716.2024718

[Bolsens, 2011] I. Bolsens, "2.5D ICs: Just a stepping stone or a long term alternative to 3D?" *Keynote Talk at 3-D Architectures for Semiconductor Integration and Packaging Conference*, 2011.

[Bordawekar, 2014] R. Bordawekar, "Analyzing analytics," https://www.computer.org/cms/Computer.org/ComputingNow/education/analyzing-analytics.pdf, Jun 2014.

[Bordawekar et al., 2014] R. Bordawekar, B. Blainey, und C. Apte, "Analyzing analytics," *SIGMOD Rec.*, Vol. 42, Nr. 4, S. 17–28, feb 2014. [Online]. Available: http://doi.acm.org/10.1145/2590989.2590993

[Bose, 2012] P. Bose, "The power of communication - trends, challenges (and accounting issues)," Feb 2012. [Online]. Available: DiscussionasNSFWETIWorkshop

[BP, 2015] BP, "Bp technology outlook," http://www.bp.com/content/dam/bp/pdf/technology/bp-technology-outlook.pdf, November 2015.

[Brockman et al., 2004] J. B. Brockman, S. Thoziyoor, S. K. Kuntz, und P. M. Kogge, "A low cost, multithreaded processing-in-memory system," in *Proceedings of the 3rd Workshop on Memory Performance Issues: In Conjunction with the 31st International Symposium on Computer Architecture*, Serie WMPI '04. New York, NY, USA: ACM, 2004, S. 16–22. [Online]. Available: http://doi.acm.org/10.1145/1054943.1054946

[Buchty, 2011] R. Buchty, "Systemaufbau: Speichertechnik," Grundlagen von Rechnerarchitekturen, Eberhard-Karls-Universität Tübingen, November 2011.

[Burton et al., 2014] E. A. Burton, G. Schrom, F. Paillet, J. Douglas, W. J. Lambert, K. Radhakrishnan, und M. J. Hill, "Fivr –- fully integrated voltage regulators on 4th generation intel^{-} core$^{\text{TM}}$ socs," in *Applied Power Electronics Conference and Exposition (APEC), 2014 Twenty-Ninth Annual IEEE*, March 2014, S. 432–439.

[Bush et al., 2016] J. Bush, M. A. Khasawneh, K. Z. Mahmoud, und T. N. Miller, "Nyuziraster: Optimizing rasterizer performance and energy in the nyuzi open source gpu," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2016, S. 204–213.

[Bush et al., 2015] J. Bush, P. Dexter, T. N. Miller, und A. Carpenter, "Nyami: a synthesizable gpu architectural model for general-purpose and graphics-specific workloads." in *ISPASS*, 2015, S. 173–182.

[Carrington et al., 2008] L. Carrington, D. Komatitsch, M. Laurenzano, M. Tikir, D. Michéa, N. Le Goff, A. Snavely, und J. Tromp, "High-frequency simulations of global seismic wave propagation using SPECFEM3D_GLOBE on 62 thousand processor cores," in *Proceedings of the SC'08 ACM/IEEE conference on Supercomputing*, I. Press, Hrsg., Austin, Texas, USA, 2008, S. 60:1–60:11.

[Carvalho de Melo, 2010] A. Carvalho de Melo, "The new linux 'perf' tools," in *Linux Kongress*, 2010.

[CEA List institute, 2011] CEA List institute, "UNISIM Virtual Platforms," http://unisim-vp.org, 2011.

[Chandrasekar et al., 2011] K. Chandrasekar, C. Weis, Y. Li, S. Goossens, M. Jung, O. Naji, B. Akesson, N. Wehn, und K. Goossens, "Drampower: Open-source dram power & energy estimation tool," 2011. [Online]. Available: http://www.drampower.info

[Chang et al., 2013] D. W. Chang, G. Byun, H. Kim, M. Ahn, S. Ryu, N. S. Kim, und M. Schulte, "Reevaluating the latency claims of 3d stacked memories," in *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific*, Jan 2013, S. 657–662.

[Chang und Kim, 2016] K. Chang und Y. Kim, "Ramulator#: A fast and lightweight DRAM simulator," 2016. [Online]. Available: https://github.com/CMU-SAFARI/RamulatorSharp

[Chatterjee et al., 2012] N. Chatterjee, R. Balasubramonian, M. Shevgoor, S. Pugsley, A. Udipi, A. Shafiee, K. Sudan, M. Awasthi, und Z. Chishti, "Usimm: the utah simulated memory module," University of Utah and Intel Corp., February 2012.

[Chen et al., 2012] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman, und N. P. Jouppi, "Cacti-3dd: Architecture-level modeling for 3d die-stacked dram main memory," in *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2012, S. 33–38.

[Choe, 2017] J. Choe, "Intel 3D XPoint Memory Die Removed from Intel Optane PCM (Phase Change Memory)," http://www.techinsights.com/about-techinsights/overview/blog/intel-3D-xpoint-memory-die-removed-from-intel-optane-pcm/, May 2017.

[Chua, 1971] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on Circuit Theory*, Vol. 18, Nr. 5, S. 507–519, Sep 1971.

[COBHAM, 1997] COBHAM, "GRLIB IP Library," http://www.gaisler.com/index.php/products/ipcores/soclibrary, 1997.

[Collange et al., 2010] S. Collange, M. Daumas, D. Defour, und D. Parello, "Barra: A Parallel Functional Simulator for GPGPU," in *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, Aug 2010, S. 351–360.

[Cooper-Balis et al., 2012] E. Cooper-Balis, P. Rosenfeld, und B. Jacob, "Buffer-on-board memory systems," in *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*, June 2012, S. 392–403.

[Cooper-Balis, 2012] E. Cooper-Balis, "Buffer-on-board memory system," Ph.D. dissertation, University of Maryland, 2012.

[Coughlin, 2016] T. Coughlin, "Crossing the chasm to new solid-state storage architectures," *IEEE Consumer Electronics Magazine*, Vol. 5, Nr. 1, S. 133–142, Jan 2016.

[Cristal et al., 2004] A. Cristal, D. Ortega, J. Llosa, und M. Valero, "Out-of-order commit processors," in *Software, IEE Proceedings-*, Feb 2004, S. 48–59.

[Cristal et al., 2005] A. Cristal, O. J. Santana, F. Cazorla, M. Galluzzi, T. Ramirez, M. Pericas, und M. Valero, "Kilo-instruction processors: overcoming the memory wall," *Micro, IEEE*, Vol. 25, Nr. 3, S. 48–57, May 2005.

[Dagum und Menon, 1998] L. Dagum und R. Menon, "Openmp: An industry-standard api for shared-memory programming," *IEEE Comput. Sci. Eng.*, Vol. 5, Nr. 1, S. 46–55, jan 1998. [Online]. Available: https://doi.org/10.1109/99.660313

[Datta, 2009] K. Datta, "Auto-tuning Stencil Codes for Cache-based Multicore Platforms," Ph.D. dissertation, University of California at Berkeley, Berkeley, CA, USA, 2009, aAI3411221.

[Davidson et al., 2006] G. S. Davidson, K. W. Boyack, R. A. Zacharski, S. C. Helmreich, und C. J. R. , *Data-centric Computing with the Netezza Architecture*, Serie Sanda Technical Report. United States. Department of Energy, April 2006.

[Davis et al., 2005] W. R. Davis, J. Wilson, S. Mick, J. Xu, H. Hua, C. Mineo, A. M. Sule, M. Steer, und P. D. Franzon, "Demystifying 3d ics: the pros and cons of going vertical," *Design Test of Computers, IEEE*, Vol. 22, Nr. 6, S. 498–510, Nov 2005.

[Deering et al., 1994a] M. F. Deering, M. G. Lavelle, und S. A. Schlapp, "A cached vram for 3d graphics," *HotChips VI*, 1994. [Online]. Available: HotChipsVI

[Deering et al., 1994b] M. F. Deering, S. A. Schlapp, und M. G. Lavelle, "Fbram: A new form of memory optimized for 3d graphics," in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, Serie SIGGRAPH '94. New York, NY, USA: ACM, 1994, S. 167–174. [Online]. Available: http://doi.acm.org/10.1145/192161.192194

[Dennard, 1968] R. H. Dennard, "Field-effect transistor memory," jun 4 1968, uS Patent 3,387,286. [Online]. Available: https://www.google.com/patents/US3387286

[Deo, 2015] M. Deo, "Enabling Next-Generation Platforms Using Altera's 3D System-in-Package Technology," Intel, White Paper, June 2015.

[Deo et al., 2016] M. Deo, J. Schulz, und L. Brown, "Intel Stratix 10 MX Devices Solve the Memory Bandwidth Challenge," Intel, White Paper, May 2016.

[DeVito et al., 2013] Z. DeVito, J. Hegarty, A. Aiken, P. Hanrahan, und J. Vitek, "Terra: A multi-stage language for high-performance computing," in *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, Serie PLDI '13. New York, NY, USA: ACM, 2013, S. 105–116. [Online]. Available: http://doi.acm.org/10.1145/2491956.2462166

[Diamos et al., 2010] G. F. Diamos, A. R. Kerr, S. Yalamanchili, und N. Clark, "Ocelot: A dynamic optimization framework for bulk-synchronous applications in heterogeneous systems," in *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, Serie PACT '10. New York, NY, USA: ACM, 2010, S. 353–364. [Online]. Available: http://doi.acm.org/10.1145/1854273.1854318

[Ding, 2017] C. Ding, "Locality Wall: Hardware Limitations and Software Opportunities," Proceedings of the Concurrent Collections Workshop, CNC, Oct. 2017, Oct 2017.

[Ding et al., 2014] J.-H. Ding, W. Hsu, B. Jeng, S. Hung, und Y. Chung, "Hsaemu - a full system emulator for hsa platforms," in *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2014 International Conference on*, Oct 2014, S. 1–10.

[Dlugosch et al., 2014] P. Dlugosch, D. Brown, P. Glendenning, M. Leventhal, und H. Noyes, "An efficient and scalable semiconductor architecture for parallel automata processing," *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, Vol. 25, Nr. 12, S. 3088–3098, Dec 2014.

[Doller, 2016] E. Doller, "Next generation memory: A competitive weapon for mobile devices," http://eecatalog.com/chipdesign/2016/07/11/next-generation-memory-a-competitive-weapon-for-mobile-devices/, July 2016, doller Consulting.

[Dong et al., 2012] X. Dong, C. Xu, Y. Xie, und N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 31, Nr. 7, S. 994–1007, July 2012.

[Dysart et al., 2016] T. Dysart, P. Kogge, M. Deneroff, E. Bovell, P. Briggs, J. Brockman, K. Jacobsen, Y. Juan, S. Kuntz, R. Lethin, J. McMahon, C. Pawar, M. Perrigo, S. Rucker, J. Ruttenberg, M. Ruttenberg, und S. Stein, "Highly scalable near memory processing with migrating threads on the emu system architecture," in *2016 6th Workshop on Irregular Applications: Architecture and Algorithms (IA3)*, Nov 2016, S. 2–9.

[Easton, 2013] J. Easton, "In-Memory Computing - Next generation technologies," https://www-01.ibm.com/events/wwe/grp/grp024.nsf/vLookupPDFs/In-Memory%20Nov%2013%202013%20John%20Easton%20Conclusions/$file/In-Memory%20Nov%2013%202013%20John%20Easton%20Conclusions.pdf, November 2013, International Business Machines Corporation.

[Egawa et al., 2013] R. Egawa, M. Sato, J. Tada, und H. Kobayashi, "Vertically integrated processor and memory module design for vector supercomputers," in *3DIC*, 2013, S. 1–6. [Online]. Available: http://dblp.org/db/conf/3dic/3dic2013.html#EgawaSTK13

[Elliott et al., 1992] D. G. Elliott, W. M. Snelgrove, und M. Stumm, "Computational ram: A memory-simd hybrid and its application to dsp," in *Custom Integrated Circuits Conference, 1992., Proceedings of the IEEE 1992*, May 1992, S. 30–6.

[Elliott et al., 1999] D. G. Elliott, M. Stumm, W. M. Snelgrove, C. Cojocaru, und R. McKenzie, "Computational ram: implementing processors in memory," *Design Test of Computers, IEEE*, Vol. 16, Nr. 1, S. 32–41, Jan 1999.

[Ellson et al., 2001] J. Ellson, E. Gansner, L. Koutsofios, S. North, G. Woodhull, S. Description, und L. Technologies, "Graphviz — open source graph drawing tools," in *Lecture Notes in Computer Science*. Springer-Verlag, 2001, S. 483–484.

[ENI, 2018] ENI, "Eni boots up hpc4 and makes its computing system the world's most powerful in the industry," https://www.eni.com/en_IT/media/2018/01/eni-boots-up-hpc4-and-makes-its-computing-system-the-worlds-most-powerful-in-the-industry, Jan 2018.

[Etgen und Dellinger, 1989] J. T. Etgen und J. Dellinger, "Accurate wave-equation modeling," in *SEG Technical Program Expanded Abstracts 1989*. Society of Exploration Geophysicists, 1989, S. 494–497.

[ETP4HPC, 2017] ETP4HPC, "European Multi-annual HPC Technology Roadmap - Strategic Research Agenda 2017," ETP4HPC - European Technology Platform For High Performance Computing, Tech. Rep., Nov 2017.

[European Exascale Projects (FP7), 2016] European Exascale Projects (FP7), "Europe towards exascale - a lookback on 5 years of european exascale research collaboration," http://exascale-projects.eu, 2016.

[Fang et al., 2007] Z. Fang, L. Zhang, J. B. Carter, A. Ibrahim, und M. A. Parker, "Active memory operations," in *Proceedings of the 21st Annual International Conference on Supercomputing*, Serie ICS '07. New York, NY, USA: ACM, 2007, S. 232–241. [Online]. Available: http://doi.acm.org/10.1145/1274971.1275004

[Fang et al., 2012] Z. Fang, L. Zhang, J. B. Carter, S. A. McKee, A. Ibrahim, M. A. Parker, und X. Jiang, "Active memory controller," *The Journal of Supercomputing*, Vol. 62, Nr. 1, S. 510–549, 2012. [Online]. Available: http://dx.doi.org/10.1007/s11227-011-0735-9

[Farmahini-Farahani et al., 2015] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, und N. S. Kim, "Drama: An architecture for accelerated processing near memory," *Computer Architecture Letters*, Vol. 14, Nr. 1, S. 26–29, Jan 2015.

[Fitch et al., 2009] B. G. Fitch, A. Rayshubskiy, M. C. Pitman, T. J. C. Ward, und R. S. Germain, "Using the active storage fabrics model to address petascale storage challenges," in *Proceedings of the 4th Annual Workshop on Petascale Data Storage*, Serie PDSW '09. New York, NY, USA: ACM, 2009, S. 47–54. [Online]. Available: http://doi.acm.org/10.1145/1713072.1713086

[Fuchs und Poulton, 1981] H. Fuchs und J. Poulton, "Pixel-planes: A vlsi-oriented design for a raster graphics engine," in *VLSI-DESIGN*, Serie 81(3), 1981, S. 20–28.

[Furber, 2016] S. B. Furber, "Brain-inspired computing," *IET Computers & Digital Techniques*, Vol. 10, Nr. 6, S. 299–305, November 2016. [Online]. Available: http://digital-library.theiet.org/content/journals/10.1049/iet-cdt.2015.0171

[Gao et al., 2015] M. Gao, G. Ayers, und C. Kozyrakis, "Practical near-data processing for in-memory analytics frameworks," in *Proceedings of the 24th International Conference on Parallel Architectures and Compilation*, Serie PACT '15. New York, NY, USA: ACM, 2015.

[Gara, 2012] A. Gara, "The long term impact of codesign," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, Nov 2012, S. 2212–2246.

[Gokhale et al., 1995] M. Gokhale, B. Holmes, und K. Iobst, "Processing in memory: The terasys massively parallel pim array," *Computer*, Vol. 28, Nr. 4, S. 23–31, Apr 1995. [Online]. Available: http://dblp.org/db/journals/computer/computer28.html#GokhaleHI95

[Gremm, 2011] A. Gremm, "Acceleration, clustering, and performance evaluation of seismic applications," Master's thesis, Eberhard-Karls-Universität Tübingen, June 2011.

[Grosser et al., 2011] T. Grosser, A. Gremm, S. Veith, G. Heim, W. Rosenstiel, V. Medeiros, und M. Eusebio de Lima, "Exploiting heterogeneous computing platforms by cataloging best solutions for resource intensive seismic applications," in *INTENSIVE 2011, The Third International Conference on Resource Intensive Applications and Services*, 2011, S. 30–36.

[Gustafson, 1988] J. L. Gustafson, "Reevaluating amdahl's law," *Communications of the ACM*, Vol. 31, S. 532–533, 1988.

[Hajkazemi et al., 2015] M. H. Hajkazemi, M. K. Tavana, und H. Homayoun, "Wide i/o or lpddr?: Exploration and analysis of performance, power and temperature trade-offs of emerging dram technologies in embedded mpsocs," in *Computer Design (ICCD), 2015 33rd IEEE International Conference on*, Oct 2015, S. 62–69.

[Hammarlund et al., 2014] P. Hammarlund, A. J. Martinez, A. A. Bajwa, D. L. Hill, E. Hallnor, H. Jiang, M. Dixon, M. Derr, M. Hunsaker, R. Kumar, R. B. Osborne, R. Rajwar, R. Singhal, R. D'Sa, R. Chappell, S. Kaushik, S. Chennupaty, S. Jourdan, S. Gunther, T. Piazza, und T. Burton, "Haswell: The fourth-generation intel core processor," *Micro, IEEE*, Vol. 34, Nr. 2, S. 6–20, Mar 2014.

[Handy, 2017] J. Handy, "Intel optane a poor fit for pcs," http://www.eetimes.com/author.asp?section_id=36&doc_id=1331116, January 2017.

[Hansson et al., 2014] A. Hansson, N. Agarwal, A. Kolli, T. F. Wenisch, und A. N. Udipi, "Simulating dram controllers for future system architecture exploration," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2014, Monterey, CA, USA, March 23-25, 2014*, 2014, S. 201–210. [Online]. Available: http://dx.doi.org/10.1109/ISPASS.2014.6844484

[Haring et al., 2012] R. A. Haring, M. Ohmacht, T. W. Fox, M. K. Gschwind, D. L. Satterfield, K. Sugavanam, P. W. Coteus, P. Heidelberger, M. A. Blumrich, R. W. Wisniewski, A. Gara, G. L. T. Chiu, P. A. Boyle, N. H. Chist, und C. Kim, "The ibm blue gene/q compute chip," *Micro, IEEE*, Vol. 32, Nr. 2, S. 48–60, March 2012.

[Hemsoth, 2015] N. Hemsoth, "The tiny chip that could disrupt exascale computing," March 2015, http://www.nextplatform.com/2015/03/12/the-little-chip-that-could-disrupt-exascale-computing/. [Online]. Available: TheNextPlatform

[Hemsoth, 2018] N. Hemsoth, "Oil and Gas Industry Gets GPU, Deep Learning Injection," https://www.nextplatform.com/2018/01/29/oil-gas-industry-gets-gpu-deep-learning-injection/amp/, Jan 2018.

[Hemsoth, 2017a] N. Hemsoth, "Exascale on the far horizon for cash-strapped oil and gas," https://www.nextplatform.com/2017/06/23/exascale-far-horizon-cash-strapped-oil-gas/, Jun 2017.

[Hemsoth, 2017b] N. Hemsoth, "Stretching the business of tape storage to extreme scale," https://www.nextplatform.com/2017/06/14/stretching-business-tape-storage-extreme-scale/, June 2017.

[Hennessy und Patterson, 2006] J. L. Hennessy und D. A. Patterson, *Computer Architecture, Fourth Edition: A Quantitative Approach.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.

[Hennessy und Patterson, 2011] J. L. Hennessy und D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*, 5. Aufl. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[Henning, 2006] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, Vol. 34, Nr. 4, S. 1–17, sep 2006. [Online]. Available: http://doi.acm.org/10.1145/1186736.1186737

[Hewlett Packard Enterprise, 2016] Hewlett Packard Enterprise, "HPE Demonstrates World's First Memory-Driven Computing Architecture," https://news.hpe.com/hewlett-packard-enterprise-demonstrates-worlds-first-memory-driven-computing-architecture/, Nov 2016.

[Hruska, 2015] J. Hruska, "Beyond ddr4: The differences between wide i/o, hbm, and hybrid memory cube," Jan 2015, extremTech. [Online]. Available: Online

[Hybrid Memory Cube Consortium, 2014] Hybrid Memory Cube Consortium, "Hybrid memory cube specification 2.1," 2014.

[IBM, 2015] IBM, "Ibm research sets new record for tape storage," https://www-03.ibm.com/press/us/en/pressrelease/46554.wss, April 2015.

[InsideHPC, 2017] InsideHPC, "China to develop exascale prototype in 2017," https://insidehpc.com/2017/01/china-develop-exascale-prototype-2017/, Jan 2017.

[Intel, 2017] Intel, "Intel Introduces the World's Most Responsive Data Center Solid State Drive," https://newsroom.intel.com/news/intel-introduces-worlds-most-responsive-data-center-solid-state-drive/, Shenzhen, March 2017.

[Ishikawa, 2015] Y. Ishikawa, "Flagship 2020 project," http://www.aics.riken.jp/aicssite/wp-content/uploads/2017/05/Flagship_2020_Project_2015.pdf, 2015.

[ISSCC, 2017] ISSCC, "ISSCC 2017 Trends," http://isscc.org/wp-content/uploads/2017/05/ISSCC2017_TechTrends.pdf, 2017.

[ITRS committee, 2014] ITRS committee, "Itrs 2.0 system integration whitepaper," Interational Technology Roadmap For Semiconductors, Tech. Rep., Dec 2014.

[Jacob, 2016] B. Jacob, "Exascale Begins at the Memory System," Gleneden Beach OR, Spring 2016.

[Jang et al., 2015] J. Jang, H. Wang, E. Kwon, J. Lee, und N. Kim, "Workload-aware optimal power allocation on single-chip heterogeneous processors," *Parallel and Distributed Systems, IEEE Transactions on*, Vol. PP, Nr. 99, S. 1–1, 2015.

[Jeddeloh und Keeth, 2012] J. Jeddeloh und B. Keeth, "Hybrid memory cube new dram architecture increases density and performance," in *VLSI Technology (VLSIT), 2012 Symposium on*, June 2012, S. 87–88.

[Jeon und Chung, 2016] D. I. Jeon und K. S. Chung, "Cashmc: A cycle-accurate simulator for hybrid memory cube," *IEEE Computer Architecture Letters*, Vol. PP, Nr. 99, S. 1–1, 2016.

[Jeong et al., 2012] M. K. Jeong, D. H. Yoon, und M. Erez, "Drsim: A platform for flexible dram system research," 2012, technical Report. [Online]. Available: http://lph.ece.utexas.edu/public/DrSim

[Jothi et al., 2011] K. Jothi, M. Sharafeddine, und H. Akkary, "Simultaneous continual flow pipeline architecture," in *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, Oct 2011, S. 127–134.

[Kagi et al., 1996] A. Kagi, J. R. Goodman, und D. Burger, "Memory bandwidth limitations of future microprocessors," in *Computer Architecture, 1996 23rd Annual International Symposium on*, May 1996, S. 78–78.

[Kahle, 2005] J. Kahle, "The Cell Processor Architecture," in *Microarchitecture, 2005. MICRO-38. Proceedings. 38th Annual IEEE/ACM International Symposium on*, Nov 2005, S. 3–3.

[Kahle, 2017] J. Kahle, "IBM Data Centric Systems Directions," Houston, TX, United States, 2017.

[Kang et al., 1999] Y. Kang, W. Huang, S.-M. Yoo, D. Keen, Z. Ge, V. Lam, P. Pattnaik, und J. Torrellas, "Flexram: toward an advanced intelligent memory system," in *Computer Design, 1999. (ICCD '99) International Conference on*, 1999, S. 192–201.

[Karp und Flatt, 1990] A. H. Karp und H. P. Flatt, "Measuring parallel processor performance," *Commun. ACM*, Vol. 33, Nr. 5, S. 539–543, may 1990. [Online]. Available: http://doi.acm.org/10.1145/78607.78614

[Kautz, 1969] W. H. Kautz, "Cellular logic-in-memory arrays," *Computers, IEEE Transactions on*, Vol. C-18, Nr. 8, S. 719–727, Aug 1969.

[Kaxiras et al., 1997] S. Kaxiras, R. Sugumar, und J. Schwarzmeier, "Distributed vector architecture: Beyond a single vector-iram," in *In First Workshop on Mixing Logic and DRAM: Chips that Compute and Remember*, 1997.

[Keable, 2012] C. Keable, "Data centric deep computing (dc2)," 2012.

[Kelly et al., 1976] K. R. Kelly, R. W. Ward, S. Treitel, und R. M. Alford, "Synthetic seismograms: A finite-difference approach," *GEOPHYSICS*, Vol. 41, Nr. 1, S. 2–27, 1976. [Online]. Available: http://dx.doi.org/10.1190/1.1440605

[Khurshid und Lipasti, 2013] M. J. Khurshid und M. Lipasti, "Data compression for thermal mitigation in the hybrid memory cube," in *Computer Design (ICCD), 2013 IEEE 31st International Conference on*, Oct 2013, S. 185–192.

[Kim et al., 2012] H. Kim, J. Lee, N. B. Lakshminarayana, J. Sim, J. Lim, und T. Pho, "Macsim: A cpu-gpu heterogeneous simulation framework," Georgia Institute of Technology, Tech. Rep., 2012.

[Kim et al., 2016] Y. Kim, W. Yang, und O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer Architecture Letters*, Vol. 15, Nr. 1, S. 45–49, Jan 2016.

[Kim und Song, 2014] Y. Kim und Y. H. Song, "Analysis of thermal behavior for 3d integration of dram," in *Consumer Electronics (ISCE 2014), The 18th IEEE International Symposium on*, June 2014, S. 1–2.

[Kim et al., 1997] Y. Kim, T.-D. Han, S.-D. Kim, und S.-B. Yang, "An effective memory-processor integrated architecture for computer vision," in *Parallel Processing, 1997., Proceedings of the 1997 International Conference on*, Aug 1997, S. 266–269.

[Kleiner et al., 1996] M. B. Kleiner, S. A. Kuhn, P. Ramm, und W. Weber, "Performance improvement of the memory hierarchy of risc-systems by application of 3-d technology," *Components, Packaging, and Manufacturing Technology, Part B: Advanced Packaging, IEEE Transactions on*, Vol. 19, Nr. 4, S. 709–718, Nov 1996.

[Knittel und Schilling, 1995] G. Knittel und A. Schilling, "Eliminating the z-buffer bottleneck," in *European Design and Test Conference, 1995. ED TC 1995, Proceedings.*, Mar 1995, S. 12–16.

[Knittel et al., 1996] G. Knittel, A. Schilling, und W. Straßer, *High Performance Computing for Computer Graphics and Visualisation: Proceedings of the International Workshop on High Performance Computing for Computer Graphics and Visualisation, Swansea 3–4 July 1995.* London: Springer London, 1996, ch. GRAMMY: High Performance Graphics Using Graphics Memories, S. 33–48. [Online]. Available: http://dx.doi.org/10.1007/978-1-4471-1011-8_3

[Kogge, 1994] P. M. Kogge, "Execube-a new architecture for scaleable mpps," in *Parallel Processing, 1994. Vol. 1. ICPP 1994. International Conference on*, Vol. 1, Aug 1994, S. 77–84.

[Kogge et al., 1995] P. M. Kogge, T. Sunaga, H. Miyataka, K. Kitamura, und E. Retter, "Combined dram and logic chip for massively parallel systems," in *Advanced Research in VLSI, 1995. Proceedings., Sixteenth Conference on*, Mar 1995, S. 4–16.

[Kogge et al., 2010] P. M. Kogge, P. La Fratta, und M. Vance, "Facing the exascale energy wall," in *Innovative Architecture for Future Generation High Performance (IWIA), 2010 International Workshop on*, Jan 2010, S. 51–58.

[Kogge, 2017] P. Kogge, "Data intensive computing, the 3rd wall, and the need for innovation in architecture," August 2017.

[Kogge, 2015] P. M. Kogge, *Updating the Energy Model for Future Exascale Systems.* Cham: Springer International Publishing, 2015, ch. High Performance Computing: 30th International Conference, ISC High Performance 2015, Frankfurt, Germany, July 12-16, 2015, Proceedings, S. 323–339.

[Kogge et al., 1997] P. M. Kogge, J. B. Brockman, T. Sterling, und G. Gao, "Processing in memory: Chips to petaflops," in *In Workshop on Mixing Logic and DRAM: Chips that Compute and Remember at ISCA '97*, 1997.

[Kopser und Vollrath, 2011] A. Kopser und D. Vollrath, "Overview of the next generation cray xmt," in *53rd Cray User Group meeting, CUG 2011*, Fairbanks, Alaska, 2011.

[Kugler et al., 1996] A. Kugler, G. Knittel, A. G. Schilling, und W. Straßer, "High-performance texture mapping architectures," in *Proceedings of the 6th OMI Annual Conference on Embedded Microprocessor Systems.* IOS Press, sep 1996, S. 189–198.

[Kyriazis, 2012] G. Kyriazis, "Heterogeneous System Architecture: A Technical Review," Advanced Micro Devices, 1.0, August 2012.

[Laplane et al., 2017] C. Laplane, P. Jobez, J. Etesse, N. Gisin, und M. Afzelius, "Multimode and long-lived quantum correlations between photons and spins in a crystal," *Phys. Rev. Lett.*, Vol. 118, Nr. 21, S. 210501, May 2017. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.118.210501

[Larsen und McAllister, 2001] E. S. Larsen und D. McAllister, "Fast matrix multiplies using graphics hardware," in *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, Serie SC '01. New York, NY, USA: ACM, 2001, S. 55–55. [Online]. Available: http://doi.acm.org/10.1145/582034.582089

[Lee et al., 2015] D. U. Lee, K. W. Kim, K. W. Kim, K. S. Lee, S. J. Byeon, J. H. Kim, J. H. Cho, J. Lee, und J. H. Chun, "A 1.2v 8gb 8-channel 128gb/s high-bandwidth memory (hbm) stacked dram with effective i/o test circuits," *Solid-State Circuits, IEEE Journal of*, Vol. 50, Nr. 1, S. 191–203, Jan 2015.

[Lee et al., 2014] S. Lee, T. Johnson, und E. Raman, "Feedback directed optimization of tcmalloc," in *Proceedings of the Workshop on Memory Systems Performance and Correctness*, Serie MSPC '14. New York, NY, USA: ACM, 2014, S. 3–1. [Online]. Available: http://doi.acm.org/10.1145/2618128.2618131

[Lee und Ro, 2013] S. Lee und W. W. Ro, "Parallel gpu architecture simulation framework exploiting work allocation unit parallelism," in *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*, April 2013, S. 107–117.

[Leeuwen, 2012] T. v. Leeuwen, "A parallel matrix-free framework for frequency-domain seismic modelling, imaging and inversion in matlab," Seismic Laboratory for Imaging and Modeling, Tech Report TR-EOAS-2012-5, 07 2012. [Online]. Available: https://www.slim.eos.ubc.ca/Publications/Public/TechReport/2012/vanleeuwen2012smii/vanleeuwen2012smii.pdf

[Leidel und Chen, 2014] J. D. Leidel und Y. Chen, "Hmc-sim: A simulation framework for hybrid memory cube devices," in *Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, May 2014, S. 1465–1474.

[Leidel und Chen, 2016] J. Leidel und Y. Chen, "Hmc-sim-2.0: A simulation platform for exploring custom memory cube operations," in *Sixth International Workshop on Accelerators and Hybrid Exascale Systems (AsHES)*, 2016, S. 10.

[Lewy et al., 1928] H. Lewy, K. Friedrichs, und R. Courant, "Über die partiellen Differenzengleichungen der mathematischen Physik," *Mathematische Annalen*, Vol. 100, S. 32–74, 1928.

[Little, 1961] J. D. C. Little, "A proof for the queuing formula: L= λ w," *Operations Research*, Vol. 9, Nr. 3, S. 383–387, 1961. [Online]. Available: http://www.jstor.org/stable/167570

[Liu et al., 2005] C. C. Liu, I. Ganusov, M. Burtscher, und S. Tiwari, "Bridging the processor-memory performance gap with 3d ic technology," *Design Test of Computers, IEEE*, Vol. 22, Nr. 6, S. 556–564, Nov 2005.

[Loh et al., 2013] G. Loh, N. Jayasena, M. Oskin, M. Nutter, D. Roberts, M. Meswani, D. P. Zhang, und M. Ignatowski, "A processing in memory taxonomy and a case for studying fixed-function pim," in *WoNDP: 1st Workshop on Near-Data Processing in conjunction with the 46th IEEE/ACM International Symposium on Microarchitecture (MICRO-46)*, 2013.

[Mai et al., 2000] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, und M. Horowitz, "Smart memories: a modular reconfigurable architecture," in *Computer Architecture, 2000. Proceedings of the 27th International Symposium on*, June 2000, S. 161–171.

[Malhotra et al., 2014] G. Malhotra, S. Goel, und S. R. Sarangi, "GpuTejas: A parallel simulator for GPU architectures," in *High Performance Computing (HiPC), 2014 21st International Conference on*, Dec 2014, S. 1–10.

[Marfurt, 1984] K. J. Marfurt, "Accuracy of finite-difference and finite-element modeling of the scalar and elastic wave equations," *GEOPHYSICS*, Vol. 49, Nr. 5, S. 533–549, 1984. [Online]. Available: http://dx.doi.org/10.1190/1.1441689

[Martin et al., 2006] G. S. Martin, R. Wiley, und K. J. Marfurt, "Marmousi2: An elastic upgrade for marmousi," *The Leading Edge*, Vol. 25, Nr. 2, S. 156–166, 2006. [Online]. Available: https://doi.org/10.1190/1.2172306

[Martonosi, 2014] M. Martonosi, "Power-Aware Computing: Then, Now, and Into the Future," *MICRO Career Workshop for Woman and Minorities in Computer Architecture*, 2014.

[McMechan, 1982] G. A. McMechan, "Determination of source parameters by wavefield extrapolation," *Geophysical Journal*, Vol. 71, S. 613–628, dec 1982.

[Medeiros, 2013] V. Medeiros, "fastRTM: An IDE to Support Development of the RTM Algorithm in High Performance FPGA platforms," Ph.D. dissertation, Universidade Federal de Pernambuco, 2013.

[Menon et al., 2014] J. Menon, L. De Carli, V. Thiruvengadam, K. Sankaralingam, und C. Estan, "Memory processing units," 2014.

[Merolla et al., 2014] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, und D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, Vol. 345, Nr. 6197, S. 668–673, 2014. [Online]. Available: http://science.sciencemag.org/content/345/6197/668

[Messina, 2017] P. Messina, "The exascale computing project," *Computing in Science Engineering*, Vol. 19, Nr. 3, S. 63–67, May 2017.

[Meuer et al., 2014] H. W. Meuer, E. Strohmaier, J. Dongarra, und H. D. Simon, *The TOP500: History, Trends, and Future Directions in High Performance Computing*, 1. Aufl. Chapman & Hall/CRC, 2014.

[Meyer et al., 2016] R. Meyer, J. Wagner, B. Farkas, S. Horsinka, P. Siegl, R. Buchty, und M. Berekovic, "A Scriptable Standard-Compliant Reporting and Logging Framework for SystemC," *ACM Trans. Embed. Comput. Syst.*, Vol. 16, Nr. 1, S. 6–1, oct 2016. [Online]. Available: http://doi.acm.org/10.1145/2983623

[Micron, 2016] Micron, "2016 analyst conference positioned for success," Feb 2016.

[Miller, 2015] T. Miller, "Openshader - open architecture gpu simulator and implementation," 2015.

[Minnick et al., 1966] R. C. Minnick, J. Goldberg, M. W. Green, W. H. Kautz, R. A. Short, H. S. Stone, und M. Yoeli, "Cellular arrays for logic and storage," Stanford Research Institute, Menlo Park, Calif., Tech. Rep., April 1966.

[Minnick, 1967] R. C. Minnick, "A survey of microcellular research," *J. ACM*, Vol. 14, Nr. 2, S. 203–241, apr 1967. [Online]. Available: http://doi.acm.org/10.1145/321386.321387

[Minutoli et al., 2015] M. Minutoli, S. Kuntz, A. Tumeo, und P. Kogge, "Implementing radix sort on emu 1," in *In the 3rd Workshop on Near-Data Processing (WoNDP)*, Waikiki, Hawaii, 2015.

[Mitsubishi, 1996] E. D. G. Mitsubishi, "3d-ram: Frame buffer memory for high-performance 3d graphics," Tech. Rep., 1996.

[Mittal et al., 2015] S. Mittal, M. Poremba, J. Vetter, und Y. Xie, "Exploring Design Space of 3D NVM and eDRAM Caches Using DESTINY Tool," http://ft.ornl.gov/sites/default/files/2015_MittalPorembaVetterXie_DESTINY_TechReport.pdf, 2015, Technical Report.

[Miura et al., 2013] N. Miura, Y. Koizumi, E. Sasaki, Y. Take, H. Matsutani, T. Kuroda, H. Amano, R. Sakamoto, M. Namiki, K. Usami, M. Kondo, und H. Nakamura, "A scalable 3d heterogeneous multi-core processor with inductive-coupling thruchip interface," in *Cool Chips XVI (COOL Chips), 2013 IEEE*, April 2013, S. 1–3.

[Moore, 1965] G. E. Moore, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, Vol. 86, Nr. 1, S. 82–85, Jan 1965.

[Morgan, 2016] T. P. Morgan, "Putting more brains in the network frees up compute," June 2016, http://www.nextplatform.com/2016/06/08/putting-brains-network-frees-compute/. [Online]. Available: TheNextPlatform

[Morton et al., 2011] S. Morton, H. Calandra, und J. Etgen, "What we need and what we can do with an exascale system," in *Proceedings of the Oil and Gas High Performance Computing Workshop 2011*, T. Rice University, Houston, Hrsg., http://static.og-hpc.org/Rice2011/Workshop-Presentations/OG-HPC%20PDF-4-WEB/Henri-John-Scott.pdf, 2011.

[Motoyoshi, 2009] M. Motoyoshi, "Through-silicon via (tsv)," *Proceedings of the IEEE*, Vol. 97, Nr. 1, S. 43–48, Jan 2009.

[Muller-Schloer et al., 1996] C. Muller-Schloer, F. Geerinckx, und B. Stanford-Smit, *Embedded Microprocessor Systems*, 1. Aufl. Amsterdam, The Netherlands: IOS Press, 1996.

[Muralimanohar et al., 2007] N. Muralimanohar, R. Balasubramonian, und N. P. Jouppi, "Cacti 6.0: A tool to model large caches," HP Laboratories, Chicago, Tech. Rep., Dec 2007, international Symposium on Microarchitecture.

[Nair et al., 2015] R. Nair, S. F. Antao, C. Bertolli, P. Bose, J. R. Brunheroto, T. Chen, C. Cher, C. H. A. Costa, J. Doi, C. Evangelinos, B. M. Fleischer, T. W. Fox, D. S. Gallo, L. Grinberg, J. A. Gunnels, A. C. Jacob, P. Jacob, H. M. Jacobson, T. Karkhanis, C. Kim, J. H. Moreno, J. K. O'Brien, M. Ohmacht, Y. Park, D. A. Prener, B. S. Rosenburg, K. D. Ryu, O. Sallenave, M. J. Serrano, P. D. M. Siegl, K. Sugavanam, und Z. Sura, "Active Memory Cube: A processing-in-memory architecture for exascale systems," *IBM Journal of Research and Development*, Vol. 59, Nr. 2/3, S. 17–1, March 2015. [Online]. Available: http://doi.acm.org/10.1147/JRD.2015.2409732

[Newman, 2010] G. A. Newman, "Large scale computing requirements for basic energy sciences - 3d geophysical modeling and imaging," BES / ASCR / NERSC Workshop, Feb 2010.

[Nvidia, 2016] Nvidia, "Nvidia Tesla P100 - The Most Advanced Datacenter Accelerator Ever Built Featuring Pascal GP100, the World's Fastest GPU," https://www.aspsys.com/images/solutions/hpc-processors/nvidia-gpu/GP100%20Pascal%20Whitepaper.pdf, 2016, Whitepaper.

[Ofenbeck et al., 2014] G. Ofenbeck, R. Steinmann, V. C. Cabezas, D. G. Spampinato, und M. Püschel, "Applying the roofline model," in *ISPASS'14*, 2014.

[Olofsson et al., 2014] A. Olofsson, T. Nordström, und Z. Ul-Abdin, "Kickstarting high-performance energy-efficient manycore architectures with epiphany," in *Signals, Systems and Computers, 2014 48th Asilomar Conference on*, Nov 2014, S. 1719–1726.

[Olofsson et al., 2011] A. Olofsson, R. Trogan, O. Raikhman, und L. Adapteva, "A 1024-core 70 gflop/w floating point manycore microprocessor," in *Poster on 15th Workshop on High Performance Embedded Computing HPEC2011*, 2011.

[Palumbo et al., 1992] R. Palumbo, H. Saiedian, und M. Zand, "The operational semantics of an active message system," in *Proceedings of the 1992 ACM Annual Conference on Communications*, Serie CSC '92. New York, NY, USA: ACM, 1992, S. 367–375. [Online]. Available: http://doi.acm.org/10.1145/131214.131261

[Panitz und Trentmann, 2015] L. Panitz und N. Trentmann, "Laut BP gibt es noch im Jahr 2050 Öl im Überfluss," https://www.welt.de/wirtschaft/energie/article148323100/Laut-BP-gibt-es-noch-im-Jahr-2050-Oel-im-Ueberfluss.html, November 2015, welt.de.

[Patterson et al., 1997a] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, und K. Yelick, "Intelligent ram (iram): chips that remember and compute," in *Solid-State Circuits Conference, 1997. Digest of Technical Papers. 43rd ISSCC., 1997 IEEE International*, Feb 1997, S. 224–225.

[Patterson et al., 1997b] D. Patterson, K. Asanovic, A. Brown, R. Fromm, J. Golbus, B. Gribstad, K. Keeton, C. Kozyrakis, D. Martin, S. Perissakis, R. Thomas, N. Treuhaft, und K. Yelick, "Intelligent ram (iram): the industrial setting, applications, and architectures," in *Computer Design: VLSI in Computers and Processors, 1997. ICCD '97. Proceedings., 1997 IEEE International Conference on*, Oct 1997, S. 2–7.

[Patterson et al., 1996] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, und K. Yelick, "A Case for Intelligent DRAM: IRAM," *IEEE Micro*, Vol. 2, Nr. 17, S. 11, August 1996. [Online]. Available: HotChipsVIII

[Patterson, 2006] D. A. Patterson, "Future of computer architecture," Feb 2006. [Online]. Available: BerkeleyEECSAnnualResearchSymposium(BEARS)

[Patterson, 2004] D. A. Patterson, "Latency lags bandwith," *Commun. ACM*, Vol. 47, Nr. 10, S. 71–75, oct 2004. [Online]. Available: http://doi.acm.org/10.1145/1022594.1022596

[Pawlowski, 2011] J. T. Pawlowski, "Hybrid memory cube (hmc)," August 2011. [Online]. Available: HOTCHIPS23

[Pleiter, 2015] D. Pleiter, "Hans meuer award for paper on in-memory processing," http://www.fz-juelich.de/SharedDocs/Meldungen/IAS/JSC/EN/2015/2015-07-hans-meuer-award.html?nn=1498884, July 2015.

[Pollack, 1999] F. J. Pollack, "New microarchitecture challenges in the coming generations of cmos process technologies (keynote address)(abstract only)," in *Proceedings of the 32Nd Annual ACM/IEEE International Symposium on Microarchitecture*, Serie MICRO 32. Washington, DC, USA: IEEE Computer Society, 1999, S. 2. [Online]. Available: http://dl.acm.org/citation.cfm?id=320080.320082

[Poremba et al., 2015a] M. Poremba, S. Mittal, D. Li, J. S. Vetter, und Y. Xie, "Destiny: A tool for modeling emerging 3d nvm and edram caches," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, S. 1543–1546.

[Poremba et al., 2015b] M. Poremba, T. Zhang, und Y. Xie, "Nvmain 2.0: A user-friendly memory simulator to model (non-)volatile memory systems," *IEEE Computer Architecture Letters*, Vol. 14, Nr. 2, S. 140–143, July 2015.

[Power et al., 2014] J. Power, J. Hestness, M. Orr, M. Hill, und D. Wood, "gem5-gpu: A heterogeneous cpu-gpu simulator," *Computer Architecture Letters*, Vol. 13, Nr. 1, Jan 2014. [Online]. Available: http://gem5-gpu.cs.wisc.edu

[Ranganathan, 2011] P. Ranganathan, "From microprocessors to nanostores: Rethinking data-centric systems," *Computer*, Vol. 44, Nr. 1, S. 39–48, Jan 2011.

[Ranjan, 2016] R. Y. Ranjan, "STT-MRAM: Emerging NVM," Jan 2016, avalanchetechnology.

[Rannacher, 2010] R. Rannacher, "ANALYSIS 2 (Differential- und Integralrechnung in $\mathbb{R}^n$)," Institut für Angewandte Mathematik Universität Heidelberg, Juli 2010, Vorlesungsskriptum.

[Reddaway, 1973] S. F. Reddaway, "Dap - a distributed array processor," *SIGARCH Comput. Archit. News*, Vol. 2, Nr. 4, S. 61–65, dec 1973. [Online]. Available: http://doi.acm.org/10.1145/633642.803971

[Renduchintala, 2017] V. Renduchintala, "Moore's law: Setting the record straight," Intel Technology and Manufacturing Day, March 2017.

[Riedel et al., 2001] E. Riedel, C. Faloutsos, G. A. Gibson, und D. Nagle, "Active disks for large-scale data processing," *Computer*, Vol. 34, Nr. 6, S. 68–74, jun 2001. [Online]. Available: http://dx.doi.org/10.1109/2.928624

[Rodrigues et al., 2011] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, und B. Jacob, "The Structural Simulation Toolkit," *SIGMETRICS Perform. Eval. Rev.*, Vol. 38, Nr. 4, S. 37–42, mar 2011. [Online]. Available: http://doi.acm.org/10.1145/1964218.1964225

[Rogers et al., 2009] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, und Y. Solihin, "Scaling the bandwidth wall: Challenges in and avenues for cmp scaling," *SIGARCH Comput. Archit. News*, Vol. 37, Nr. 3, S. 371–382, jun 2009. [Online]. Available: http://doi.acm.org/10.1145/1555815.1555801

[Rogers und Earnshaw, 1991] D. F. Rogers und R. Earnshaw, *State of the Art in Computer Graphics: Visualization and Modeling*, 1. Aufl. Springer Publishing Company, Incorporated, 1991.

[Rosenfeld, 2014] P. Rosenfeld, "Performance exploration of the hybrid memory cube," Ph.D. dissertation, University of Maryland, 2014, ph.D. thesis.

[Rosenfeld et al., 2011] P. Rosenfeld, E. Cooper-Balis, und B. Jacob, "DRAMSim2: A Cycle Accurate Memory System Simulator," *IEEE Computer Architecture Letters*, Vol. 10, Nr. 1, S. 16–19, Jan 2011.

[Rothbart, 2017] S. Rothbart, "Expanded use of new microscopy technology requires an innovative approach to storage," https://www.hpcwire.com/2017/04/17/expanded-use-new-microscopy-technology-requires-innovative-approach-storage, April 2017, van Andel Research Institute Center of Epigenetics.

[Sakuma et al., 2008] K. Sakuma, P. S. Andry, K. Sueoka, R. Horton, S. L. Wright, Y. Oyama, B. C. Webb, C. Patel, B. Dang, C. K. Tsang, E. Sprogis, R. Polastre, und J. U. Knickerbocker, "Die cavity integration technology for through-silicon-vias stacking," in *Manufacturing and Reliability Challenges for 3D ICs using TSVs*, San Diego, CA, 2008.

[Samsung, 2014] Samsung, "Samsung V-NAND technology," https://www.samsung.com/us/business/oem-solutions/pdfs/V-NAND_technology_WP.pdf, Sep 2014.

[SanDisk, 2015] SanDisk, "Sandisk and hp launch partnership to create memory-driven computing solutions," Oct 2015, https://www.sandisk.com/about/media-center/press-releases/2015/sandisk-and-hp-launch-partnership.

[Saulsbury et al., 1996] A. Saulsbury, F. Pong, und A. Nowatzyk, "Missing the memory wall: The case for processor/memory integration," in *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, Serie ISCA '96. New York, NY, USA: ACM, 1996, S. 90–101. [Online]. Available: http://doi.acm.org/10.1145/232973.232984

[Schling, 2011] B. Schling, *The Boost C++ Libraries*. XML Press, 2011.

[Schuster, 2014] T. Schuster, "Socrocket: eine flexible erweiterbare virtuelle plattform zum entwurf robuster eingebetteter systeme," Ph.D. dissertation, Carl-Friedrich-Gauß-Fakultät, Technische Universität Braunschweig, September 2014. [Online]. Available: https://books.google.com/books?id=YVtjjwEACAAJ

[Scrbak et al., 2015] M. Scrbak, M. Islam, K. Kavi, M. Ignatowski, und N. Jayasena, "Lecture notes in computer science: Exploring the design space," in *Architecture of Computing Systems – ARCS 2015*, L. M. P. Pinho, W. Karl, A. Cohen, und U. Brinkschulte, Hrsgg. Springer International Publishing, 2015, Vol. 9017, ch. Processing-in-Memory, S. 43–54. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-16086-3_4

[Seagate, 2016] Seagate, "Seagate demonstrates fastest-ever ssd flash drive," March 2016, http://www.seagate.com/de/de/about-seagate/news/seagate-demonstrates-fastest-ever-ssd-flash-drive-pr/. [Online]. Available: PressRelease

[Sebexen und Sohmers, 2015] P. Sebexen und T. Sohmers, "Software techniques for scratchpad memory management," in *Proceedings of the 2015 International Symposium on Memory Systems*, Serie MEMSYS '15. New York, NY, USA: ACM, 2015, S. 98–102. [Online]. Available: http://doi.acm.org/10.1145/2818950.2818966

[Sei, 1995] A. Sei, "A family of numerical schemes for the computation of elastic waves," *SIAM Journal on Scientific Computing*, Vol. 16, Nr. 4, S. 898–916, 1995. [Online]. Available: https://doi.org/10.1137/0916052

[Semiconductor, 2005] T. Semiconductor, "Tezzaron unveils 3d sram," January 2005.

[Shainer, 2016] G. Shainer, "Intelligent networks: A new co-processor emerges," March 2016, http://www.nextplatform.com/2016/03/02/intelligent-networks-a-new-co-processor-emerges/. [Online]. Available: TheNextPlatform

[Shalf und Leland, 2015] J. M. Shalf und R. Leland, "Computing beyond moore's law," *Computer*, Vol. 48, Nr. 12, S. 14–23, Dec 2015.

[Shimizu et al., 1996] T. Shimizu, J. Korematu, M. Satou, H. Kondo, S. Iwata, K. Sawai, und e. a. , "A multimedia 32b risc microprocessor with 16mb dram," in *Solid-State Circuits Conference, 1996. Digest of Technical Papers. 42nd ISSCC., 1996 IEEE International*, Feb 1996, S. 216–217.

[Siegl et al., 2015] P. Siegl, R. Buchty, und M. Berekovic, "Revealing Potential Performance Improvements By Utilizing Hybrid Work-Sharing For Resource-Intensive Seismic

Applications," in *Proceedings of the 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2015, Turku, Finland, March 4-6, 2015*. IEEE Computer Society, Mar 2015. [Online]. Available: http://dx.doi.org/10.1109/PDP.2015.28

[Siegl et al., 2016a] P. Siegl, R. Buchty, und M. Berekovic, "Data-Centric Computing Frontiers: A Survey On Processing-In-Memory," in *Proceedings of the Second International Symposium on Memory Systems, MEMSYS 2016, Washington, DC, USA, October 3-6, 2016*. ACM, Oct 2016, S. 295–308. [Online]. Available: http://doi.acm.org/10.1145/2989081.2989087

[Siegl et al., 2016b] P. Siegl, R. Buchty, und M. Berekovic, "Towards Bridging the Gap Between Academic and Industrial Heterogeneous System Architecture Design Space Exploration," in *Proceedings of the 2016 Workshop on Rapid Simulation and Performance Evaluation - Methods and Tools, RAPIDO@HiPEAC 2016, Prague, Czech Republic, January 18, 2016*. ACM, Jan 2016, S. 1–6. [Online]. Available: http://doi.acm.org/10.1145/2852339.2852343

[Siegl, 2012] P. Siegl, "Hybride Beschleunigung einer seismischen Applikation durch Kombination von traditionellen Methoden und OpenCL," Eberhard Karls Universität, Tübingen, 2012, Diplomarbeit.

[Smith, 1995] B. Smith, "Computer architectures," *ARPA Principal Investigator meeting notes*, 1995.

[Sodani, 2011] A. Sodani, "Race to exascale: Opportunities and challenges," in *Keynote at the Annual IEEE/ACM 44th Annual International Symposium on Microarchitecture*, 2011.

[Stanley-Marbell et al., 2011] P. Stanley-Marbell, V. C. Cabezas, und R. P. Luijten, "Pinned to the walls - impact of packaging and application properties on the memory and power walls," in *Low Power Electronics and Design (ISLPED) 2011 International Symposium on*, Aug 2011, S. 51–56.

[Starke et al., 2015] W. J. Starke, J. Stuecheli, D. M. Daly, J. S. Dodson, F. Auernhammer, P. M. Sagmeister, G. L. Guthrie, C. F. Marino, M. Siegel, und B. Blaner, "The cache and memory subsystems of the ibm power8 processor," *IBM Journal of Research and Development*, Vol. 59, Nr. 1, S. 3–1, Jan 2015.

[Stockwell, 2011] J. W. Stockwell, "Geophysical image processing with seismic unix: Gpgn 461/561 lab," enter for Wave Phenomena, Colorado School of Mines, Golden, Colorado, Tech. Rep., nov 2011. [Online]. Available: http://www.cwp.mines.edu/~john/GPGN461.561/index.html

[Stone, 1970] H. S. Stone, "A logic-in-memory computer," *Computers, IEEE Transactions on*, Vol. C-19, Nr. 1, S. 73–78, Jan 1970.

[Stratton et al., 2012] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, und W.-m. W. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," in *IMPACT Technical Report*. University of Illinois at Urbana-Champaign, 2012.

[Thorbecke, 2017] J. Thorbecke, "2d finite-difference wavefield modelling," https://janth.home.xs4all.nl/Software/fdelmodcManual.pdf, March 2017.

[Thorolfsson et al., 2009] T. Thorolfsson, N. Moezzi-Madani, und P. D. Franzon, "A low power 3d integrated fft engine using hypercube memory division," in *ISLPED*, Serie ISLPED '09. New York, NY, USA: ACM, 2009, S. 231–236. [Online]. Available: http://dblp.org/db/conf/islped/islped2009.html#ThorolfssonMF09

[Thoziyoor et al., 2005] S. Thoziyoor, J. Brockman, und D. Rinzler, "Pim lite: A multithreaded processor-in-memory prototype," in *Proceedings of the 15th ACM Great Lakes Symposium on VLSI*, Serie GLSVLSI '05. New York, NY, USA: ACM, 2005, S. 64–69. [Online]. Available: http://doi.acm.org/10.1145/1057661.1057678

[Toshiba, 2017] Toshiba, "Toshiba memory corporation develops world's first 3d flash memory with tsv technology," https://toshiba.semicon-storage.com/de/company/news/2017/07/memory-20170711-1.html, July 2017.

[Trader, 2015] T. Trader, "Mellanox touts arrival of intelligent interconnect," November 2015, http://www.hpcwire.com/2015/11/16/mellanox-touts-arrival-of-intelligent-interconnect/. [Online]. Available: HPCwire

[Tsuboi et al., 2016] S. Tsuboi, K. Ando, T. Miyoshi, D. Peter, D. Komatitsch, und J. Tromp, "A 1.8 trillion degrees-of-freedom, 1.24 petaflops global seismic wave simulation on the k computer," *The International Journal of High Performance Computing Applications*, Vol. 30, Nr. 4, S. 411–422, 2016. [Online]. Available: http://dx.doi.org/10.1177/1094342016632596

[TU Delft, TNO and DELPHI, 2002] TU Delft, TNO and DELPHI, "Subsalt Multiples Attenuation And Reduction Team (SMAART)," http://www.delphi.tudelft.nl/SMAART, 2002, joint Venture consisting of BHP Billiton, BP and ChevronTexaco. [Online]. Available: http://www.delphi.tudelft.nl/SMAART

[Ubal et al., 2012] R. Ubal, B. Jang, P. Mistry, D. Schaa, und D. Kaeli, "Multi2sim: A simulation framework for cpu-gpu computing," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, Serie PACT '12. New York, NY, USA: ACM, 2012, S. 335–344. [Online]. Available: http://doi.acm.org/10.1145/2370816.2370865

[Valverde, 2015] D. Valverde, "Theia: ray graphic processing unit," 2015.

[Venray Technology Ldt., 2014] Venray Technology Ldt., "Tomi technology implementations," http://www.venraytechnology.com/Implementations.htm, 2014.

[Versteeg, 1994] R. Versteeg, "The marmousi experience: Velocity model determination on a synthetic complex data set," *The Leading Edge*, Vol. 13, Nr. 9, S. 927–936, 1994. [Online]. Available: http://dx.doi.org/10.1190/1.1437051

[von Neumann, 1945] J. von Neumann, "First draft of a report on the edvac," *Annals of the History of Computing, IEEE*, Vol. 15, Nr. 4, S. 27–75, 1945.

[Waldrop, 2016] M. M. Waldrop, "More than moore," *NATURE*, Vol. 530, S. 144–147, feb 2016.

[Walter, 2007] R. Walter, *Einführung in die Analysis*, Serie De Gruyter Lehrbuch. de Gruyter, 2007, Nr. Bd. 2. [Online]. Available: https://books.google.com/books?id=VA6q4IQI78AC

[Wang et al., 2005] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, und B. Jacob, "DRAMsim: A Memory System Simulator," *SIGARCH Comput. Archit. News*, Vol. 33, Nr. 4, S. 100–107, nov 2005. [Online]. Available: http://doi.acm.org/10.1145/1105734.1105748

[Wang et al., 2012] P.-H. Wang, C.-W. Lo, C.-L. Yang, und Y.-J. Cheng, "A cycle-level simt-gpu simulation framework," in *Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium on*, April 2012, S. 114–115.

[Weaver und Germond, 1994] D. L. Weaver und T. Germond, "The sparc architecture manual, version 9," SPARC International, Inc., San Jose, California, Tech. Rep., 1994.

[Weickert, 2004] J. Weickert, "Kapitel 52: Partielle Ableitungen," http://www.mia.uni-saarland.de/Teaching/MFI04/mfi2script_TeilE.pdf, Universität Saarland, 2004, Mathematik für Informatiker, WS2003/2004, SS2004 .

[Whitmore, 1983] N. D. Whitmore, *Iterative depth migration by backward time propagation*, Serie 1983 SEG Annual Meeting. SEG, 1983, S. 382–385. [Online]. Available: http://library.seg.org/doi/abs/10.1190/1.1893867

[Williams et al., 2009] S. Williams, A. Waterman, und D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, Vol. 52, Nr. 4, S. 65–76, apr 2009. [Online]. Available: http://doi.acm.org/10.1145/1498765.1498785

[Wimmer-Schweingruber, 2007] R. Wimmer-Schweingruber, "Einführung in die Physik Teil I - Vorlesung 13: Akustik," http://www.ieap.uni-kiel.de/et/people/wimmer/teaching/PhysI/V13.pdf, Christian Albrechts University Kiel, Februar 2007.

[Wolff et al., 2004] M. Wolff, P. Hauck, und W. Küchlin, *Mathematik für Informatik und BioInformatik*. Springer Berlin Heidelberg, 2004. [Online]. Available: https://books.google.com/books?id=xbKPcvVWSWgC

[Wulf und McKee, 1995] W. A. Wulf und S. A. McKee, "Hitting the memory wall: Implications of the obvious," *SIGARCH Comput. Archit. News*, Vol. 23, Nr. 1, S. 20–24, mar 1995. [Online]. Available: http://doi.acm.org/10.1145/216585.216588

[Xie, 2013] Y. Xie, "Future memory and interconnect technologies," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, March 2013, S. 964–969.

[Yoshida, 2017] J. Yoshida, "Rambus, Microsoft Heat Up With Cold DRAM," http://www.eetimes.com/document.asp?_mc=RSS_EET_EDT&doc_id=1331608, April 2017.

[Zakharenko et al., 2013] V. Zakharenko, T. Aamodt, und A. Moshovos, "Characterizing the performance benefits of fused cpu/gpu systems using fusionsim," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, March 2013, S. 685–688.

[Zhang et al., 2013] D. P. Zhang, N. Jayasena, A. Lyashevsky, J. Greathouse, M. Meswani, M. Nutter, und M. Ignatowski, "A new perspective on processing-in-memory architecture

design," in *Proceedings of the ACM SIGPLAN Workshop on Memory Systems Performance and Correctness*, Serie MSPC '13. New York, NY, USA: ACM, 2013, S. 7–1. [Online]. Available: http://doi.acm.org/10.1145/2492408.2492418

[Zhang et al., 2014] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, und M. Ignatowski, "Top-pim: Throughput-oriented programmable processing in memory," in *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, Serie HPDC '14. New York, NY, USA: ACM, 2014, S. 85–98. [Online]. Available: http://doi.acm.org/10.1145/2600212.2600213

[Zhang et al., 2015] X. Zhang, Y. Zhang, und J. Yang, "Dlb: Dynamic lane borrowing for improving bandwidth and performance in hybrid memory cube," in *2015 33rd IEEE International Conference on Computer Design (ICCD)*, Oct 2015, S. 125–132.

[Zhu et al., 2013] Q. Zhu, B. Akin, H. E. Sumbul, F. Sadi, J. C. Hoe, L. Pileggi, und F. Franchetti, "A 3d-stacked logic-in-memory accelerator for application-specific data intensive computing," in *3DIC*, Oct 2013, S. 1–7.

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AI	Arithmetische Intensität
ALU	Arithmetic Logic Unit
AMC	(IBM) Active Memory Cube
APF	Actual Pressure Field
API	Application Programming Interface
APU	Accelerated Processing Unit
ASIC	Application-Specific Integrated Circuit
ASIP	Application-Specific Instruction set Processor
BFS	Breadth-First Search
BOB	Buffer-On-Board
ccNUMA	cache-coherent NUMA
CFL	Courant-Friedrichs-Lewy condition
CGRA	Coarse-Grained Reconfigurable Architecture
CMOS	Complementary Metal-Oxide-Semiconductor
CRC	Cyclic Redundancy Check
CWP	Center For Wave Phenomena
DARPA	Defense Advanced Research Projects Agency
DCS	Data-Centric Systems
DDR	Double Data Rate
DMA	Direct Memory Access
DoE	Department Of Energy (Energieministerium der USA)
DRAM	Dynamic Random-Access Memory
DSP	Digital Signal Processor
ERAT	Effective to Real Address Translation
FIFO	First In First Out
FLIT	FLow control unIT
FMA	Fused-Multiply-Add
FLOPS	FLoating-point Operations Per Second
FPGA	Field-Programmable Gate Array
FWI	Full-Waveform Inversion
GPGPU	General-Purpose computing on GPU
GPU	Graphics Processing Unit
HBM	High-Bandwidth Memory
HMC	Hybrid Memory Cube
HPC	High-Performance Computing
HSA	Heterogeneous System Architecture
IC	Integrated Circuit

ID	Identifier
ILP	Instruction-Level Parallelism
IOPS	Integer Operations Per Second
IP	Intellectual Property
IPC	Instructions Per Cycle
ISP	Instruction-Set Architecture
JIT	Just-In-Time
LBM	Lattice-Boltzmann-Methoden
LLVM	*ehemals:* Low Level Virtual Machine
LRU	Least Recently Used
NAND	Negative-AND
NDP	Near-Data Processing
NMP	Near-Memory Processing
NP	Nondeterministic Polynomial time
NPF	Next Pressure Field
NUMA	Non-Uniform Memory Access
NVM	Non-Volatile Memory
MCDRAM	(Intel) Multi-Channel DRAM
MIC	Many Integrated Core architecture
MIMD	Multiple Instruction, Multiple Data
MMIO	Memory-Mapped Input/Output
MMU	Memory Management Unit
PC	Program Counter
PCB	Printed Circuit Board
PCM	Phase-Change Memory
PGAS	Partitioned Global Address Space
PIM	Processing-In-Memory
PLRU	Pseudo-LRU
PPF	Previous Pressure Field
ROB	Re-Order Buffer
RTL	Register-Transfer Level
RTM	Reverse-Time Migration
SCM	Storage Class Memory
SDB	Slice-And-Data Buffer
SDR	Single Data Rate
SDL	Simple DirectMedia Layer
SerDes	Serialisierer / Deserialisierer
SIMD	Single Instruction, Multiple Data
SLC	Single-Level Cell
SMB	(Intel) Scalable Memory Buffer
SoC	System-On-Chip
SRAM	Static Random-Access Memory
STL	Standard Template Library
SU	Seismic Un*x
TSV	Through-Silicon Vias
TTI	Tilted Transverse Isotropic

UMA	Uniform-Memory Access
VLIW	Very Long Instruction Word
VP	Virtual Plattform
VTI	Vertical Transverse Isotropic

Lebenslauf von Patrick D. M. Siegl

Persönliche Daten

Geboren am 14. Mai 1986 in Stuttgart (Baden-Württemberg)
Nationalität: deutsch

Schulausbildung

Eugen Bolz Gymnasium, Rottenburg am Neckar — **09/1997 – 06/2006**
Abschluss: Allgemeine Hochschulreife (Abitur)

Hochschulstudium

Eberhard Karls Universität Tübingen — **09/2006 – 03/2012**
Studium der Informatik (Nebenfach: Medienwissenschaften)
Abschluss: Diplom-Informatiker (Dipl.-Inf.)

Studienarbeit am Lehrstuhl Technische Informatik — **07/2011**
Thema: „Implementierung einer Private-Cloud-Lösung mit Libvirt und QEMU-KVM unter Linux auf System z“

Diplomarbeit am Lehrstuhl Technische Informatik — **04/2012**
Thema: „Hybride Beschleunigung einer seismischen Applikation durch Kombination von traditionellen Methoden und OpenCL“

Berufstätigkeit

Schulpraktikant bei dem IBM Global Service/Sales Center in Herrenberg — **04/2003**

Studentische Aushilfe bei der ARLT Computer GmbH in Tübingen — **06/2006 – 01/2008**

Studentische Aushilfe bei Teamwork Transport in Tübingen — **05/2008 – 03/2009**

Werkstudent Informatik / Mediendesign bei der Stellwerk3 GmbH in Reutlingen — **01/2009 – 09/2011**

Research Supplemental in der Abteilung *Exascale System Software*, IBM Thomas J. Watson Research Center in Yorktown Heights (NY, USA) — **07/2012 – 02/2014**

Wissenschaftlicher Mitarbeiter am Lehrstuhl für Technische Informatik (E.I.S.), Institut Theoretische Informatik, Technische Universität Carolo-Wilhelmina zu Braunschweig — **04/2014 – 09/2016**

Software Engineering Researcher in der Abteilung *Data Centric System Software*, IBM Thomas J. Watson Research Center in Yorktown Heights (NY, USA) — **seit 09/2016**

Eigene Publikationen

R. Meyer, J. Wagner, B. Farkas, S. Horsinka, P. Siegl, R. Buchty, und M. Berekovic, "A Scriptable Standard-Compliant Reporting and Logging Framework for SystemC," *ACM Trans. Embed. Comput. Syst.*, Vol. 16, Nr. 1, S. 6–1, oct 2016. [Online]. Available: http://doi.acm.org/10.1145/2983623

S. Michalik, R. Meyer, S. Michalik, P. Siegl, M. Berekovic, und L. Fossati, "TLM Design Space Exploration for a Hardware CFDP Transmission Accelerator," in *Proceedings of the 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. Johannes Kepler University Linz, Mar 2015.

R. Nair, S. F. Antao, C. Bertolli, P. Bose, J. R. Brunheroto, T. Chen, C. Cher, C. H. A. Costa, J. Doi, C. Evangelinos, B. M. Fleischer, T. W. Fox, D. S. Gallo, L. Grinberg, J. A. Gunnels, A. C. Jacob, P. Jacob, H. M. Jacobson, T. Karkhanis, C. Kim, J. H. Moreno, J. K. O'Brien, M. Ohmacht, Y. Park, D. A. Prener, B. S. Rosenburg, K. D. Ryu, O. Sallenave, M. J. Serrano, P. D. M. Siegl, K. Sugavanam, und Z. Sura, "Active Memory Cube: A processing-in-memory architecture for exascale systems," *IBM Journal of Research and Development*, Vol. 59, Nr. 2/3, S. 17–1, March 2015. [Online]. Available: http://doi.acm.org/10.1147/JRD.2015.2409732

P. Siegl, R. Buchty, und M. Berekovic, "Revealing Potential Performance Improvements By Utilizing Hybrid Work-Sharing For Resource-Intensive Seismic Applications," in *Proceedings of the 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2015, Turku, Finland, March 4-6, 2015*. IEEE Computer Society, Mar 2015. [Online]. Available: http://dx.doi.org/10.1109/PDP.2015.28

P. Siegl, R. Buchty, und M. Berekovic, "Data-Centric Computing Frontiers: A Survey On Processing-In-Memory," in *Proceedings of the Second International Symposium on Memory Systems, MEMSYS 2016, Washington, DC, USA, October 3-6, 2016*. ACM, Oct 2016, S. 295–308. [Online]. Available: http://doi.acm.org/10.1145/2989081.2989087

P. Siegl, R. Buchty, und M. Berekovic, "Towards Bridging the Gap Between Academic and Industrial Heterogeneous System Architecture Design Space Exploration," in *Proceedings of the 2016 Workshop on Rapid Simulation and Performance Evaluation - Methods and Tools, RAPIDO@HiPEAC 2016, Prague, Czech Republic, January 18, 2016*. ACM, Jan 2016, S. 1–6. [Online]. Available: http://doi.acm.org/10.1145/2852339.2852343

P. Siegl, R. Buchty, B. Farkas, S. A. Horsinka, R. Meyer, J. Wagner, und M. Berekovic, "The Past, Present and Future of the Open-Source Virtual Platform SoCRocket," in *Proceedings of the 2016 Workshop on Mixed Criticality Applications and Implementation Approaches, EMC^2@HiPEAC 2016, Prague, Czech Republic, January 20, 2016*, Jan 2016.

P. Siegl, R. Buchty, und M. Berekovic, “A Bandwidth Accurate, Flexible and Rapid Simulating Multi-HMC Modelling Tool,” in *Proceedings of the Third International Symposium on Memory Systems, MEMSYS 2017, Washington, DC, USA, October 2-5, 2017*. ACM, Oct 2017, S. 71–82. [Online]. Available: http://doi.acm.org/10.1145/3132402.3132403

P. Siegl, “Implementierung einer Private-Cloud-Lösung mit Libvirt und KVM unter Linux auf System z,” Eberhard Karls Universität, Tübingen, 2011, Studienarbeit.

P. Siegl, “Hybride Beschleunigung einer seismischen Applikation durch Kombination von traditionellen Methoden und OpenCL,” Eberhard Karls Universität, Tübingen, 2012, Diplomarbeit.